VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Využití ”random forestů” pro rozpoznávání objektů v obrazech

The Use of Random Forests for Recognising Objects in Images

2017

Bc. Petr Ehler

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Petr Ehler**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Využití "random forestů" pro rozpoznávání objektů v obrazech
The Use of Random Forests for Recognising Objects in Images

Jazyk vypracování: čeština

Zásady pro vypracování:

Analýza obrazu může být založena na využití pohybujícího se okna, na výpočtu příznaků a konečně na použití klasifikátoru. Častěji používanými příznaky jsou například histogram orientovaných gradientů (HOG) nebo příznaky Haarovy. Oblíbenými klasifikátory jsou například SVM klasifikátory (support vector machine), rozhodovací stromy nebo lesy rozhodovacích stromů (random forests). Cílem diplomové práce je prakticky ověřit užitečnost lesů rozhodovacích stromů (random forests).

V diplomové práci proveďte:
1. Seznamte se s teorií lesů rozhodovacích stromů. Teorii pak v textové práci také pěkně popište.
2. Zvolte objekt, na němž provedete praktické testování; může se jednat například o postavy.
3. Zvolte příznaky; můžete využít např. histogramu orientovaných gradientů.
4. Implementujte program, pomocí něhož provedete porovnání. Implementaci random forests můžete využít z dostupných knihoven.
5. Sestavte vhodnou trénovací a testovací sadu a proveďte porovnání. Výsledky podrobně zdokumentujte.

Seznam doporučené odborné literatury:

[1] Breiman, L.: Random forests. Machine learning 45: 5-32, 2001
Další dle pokynů vedoucího diplomové práce a vlastního průzkumu.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **doc. Dr. Ing. Eduard Sojka**

Datum zadání: 01.09.2015

Datum odevzdání: 14.07.2017

doc. Ing. Jan Platoš, Ph.D.
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 17. července 2017 .....

Děkuji panu doc. Dr. Ing. Eduardu Sojkovi za pomoc při vedení diplomové práce. Mé poděkování patří také ostatním, kteří mi při zpracování pomáhali.

Abstrakt

Cílem této diplomové práce je popsat využití Random forestu a jeho schopnost rozpoznání objektů v obrazech. Teoretická část se zabývá názvoslovím a postupy při sestavování rozhodovacího stromu a Random forestu. Praktická část má za úkol porovnat úspěšnost a rychlost vyhodnocování obrazu pomocí metody Random forestu a metody SVM. Následně je vytvořen program pro detekci vozidel na křižovatce.

Klíčová slova: Zpracování obrazu, Rozhodovací strom, Náhodný les, Random forest, SVM, HOG, OpenCV, C++

Abstract

The aim of this master's thesis is to describe use of Random Forest and its ability to recognize objects in images. The theoretical part deals with terminology and process of assembling decision tree and Random forest. The practical part compares success rate and duration of image classification using Random forest and SVM methods. Thereafter a program for detecting vehicles at the junction is created.

Keywords: Image processing, Decission tree, Random forrest, SVM, HOG, OpenCV, C++

Seznam použitých zkratek a symbolů

$\mathbb{N}$	–	přirozená čísla $\{1, 2, 3, ..., +\infty\}$
$\mathbb{Z}$	–	celá čísla
$\mathbb{R}^n$	–	n-rozměrný euklidovský reálný prostor
A^{-1}	–	inverzní matice k matici A
A^T	–	transponovaná matice k matici A
■	–	konec důkazu, věty nebo příkladu
RF	–	Random forest (Náhodný les)
LBP	–	Lokální binární vzor
HOG	–	Histogramy orientovaných gradientů
SVM	–	Support vector machine
ROI	–	Region Of Interest (Oblast zájmu)
px	–	Pixel
TP	–	True positive
TN	–	True negative
FP	–	False positive
FN	–	False negative
ACC	–	Accuracy, přesnost
TPR	–	True positive rate
FPR	–	False positive rate
PPV	–	Positive predictive value
NPV	–	Negative predictive value
F_1	–	F_1 Score

Obsah

Seznam tabulek

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Základním, a pravděpodobně i nejdůležitějším smyslem pro člověka, je zrak. Pomocí tohoto smyslu vnímáme a přijímáme přibližně 80% veškerých informací. Není proto žádným překvapením, že tuto schopnost se snažíme přenést do ostatních oblastí našeho života.

Této schopnosti se stále častěji snažíme naučit nejrůznější druhy informačních technologií. Rozpoznávání objektů pomocí technologie získává stále větší oblibu a uplatnění nejen v oborech, které byly vždy technologicky velmi vyspělé, jako je obrana, medicína a letectví, ale i v běžném životě každého z nás. Tento nástroj nás každodenně doprovází na cestě do práce v podobě řízených křižovatek, automatického zaostření fotoaparátu v mobilním telefonu nebo zatím jako hudba budoucnosti v podobě automatem řízených vozidel.

Je jisté, že uplatnění tohoto smyslu v IT má širší využití a napomáhá ke zjednodušení, zefektivnění a zrychlení práce, vyšší automatizaci procesů a tím druhořadě i úspoře nákladů a firmám ke zvýšení jejích zisků.

Vnímání, přijímaní a zpracování obrazu se pro člověka jeví snadnou úlohou, získanou na základě zkušeností. Z pohledu kusu informační technologie se jedná o podobný problém. Podobně jako u lidí je potřeba program nejdříve objekty naučit rozlišovat. Na trénovací množině se program naučí, jak daná skupina příznaků identifikuje daný objekt. Pokud ale příznaky vypadají jinak, může se jednat o objekt jiný. O tom, jak program naučit rozpoznávat objekty, pojednává do jisté míry právě tato diplomová práce.

Popisuje způsob a příklady zpracování objektů v obrazech využitím metody Random forestu. Mezi výhody metody RF řadíme fakt, že je flexibilní, neboť pro modelování může být použit velký počet atributů. Přitom lze algoritmus aplikovat jak na malé, tak na velké soubory dat, které mohou být snadno a poměrně rychle vyhodnoceny. Další nespornou výhodou Random forestu oproti jiným metodám je skutečnost, že pro poměrně přesné rozhodování není vyžadována dokolaná trénovací množina. Předností je i velmi krátký časový interval pro naučení algoritmu zpracovávat požadovaná data. Jako každá metoda má i tato své nedostatky. Jeden z nich je jeho pomalá předpovídací schopnost a v některých případech i náročnost na složitost jednotlivých rozhodovacích stromů.

Cílem této práce je seznámit čtenáře s metodou Random forest a její aplikací na rozpoznávání objektů v obraze. První část diplomové práce je věnována klasifikací základních pojmů, které jsou nezbytné pro pochopení daného tématu. Důraz je kladen především na sestavení rozhodovacího stromu, který je stěžejní pro aplikaci vybrané metody. V dalších

částech diplomové práce je na jednoduchém příkladě vysvětleno, jakými způsoby se dá Random forest sestavit.

Závěrečná část se věnuje porovnání úspěšnosti a rychlosti vyhodnocení obrazu metodami Random forest a SVM s kernelem Linear a kernelem RBF. Výsledky měření budou použity pro sestavení programu pro detekci vozidel projíždějících přes křižovatku, který bude schopen vyhodnocovat obrazy z kamery v reálném čase.

2 Základní pojmy

Tato kapitola je věnována vysvětlení základních pojmů, se kterými se dále pracuje. Důraz je zde kladen převážně na vysvětlení pojmů potřebných k sestavení Random forestu jako ke stěžejnímu tématu diplomové práce. Potřebné názvosloví nezbytné pro vysvětlení postupu a zpracování zadaného tématu je zde zavedeno vždy jako název jednotlivých podkapitol.

2.1 Rozhodovací strom

Smyslem rozhodovacího stromu je zařazení vstupního objektu do jedné z výsledných tříd.

V zásadě je rozhodovací strom reprezentován disjunkcemi konjunkcí stanovených podmínek nad hodnotami atributů jednotlivých instancí. Každá cesta od kořene stromu k jeho listu koresponduje s konjunkcí testů atributů a celý strom je disjunkcí těchto konjunkcí. [9]

Jinými slovy, rozhodovací strom je možné si představit jako vývojový diagram mající počátek ve výchozím uzlu (**kořenu**), kde vnitřní **uzel** představuje test na atribut. Každá z **hran** (větví), vycházející z uzlu, představuje výsledek daného testu. Koncový uzel (**list**) určuje třídu, do které byl objekt zařazen. Cesta od kořene k listu je představována klasifikačními pravidly, ve kterých se strom větví, nicméně později již nedochází k opětovnému propojovaní jednotlivých uzlů.

Formálně můžeme definici napsat jako:

Definice 2.1 *Mějme databázi* $T = \left(\vec{t}_1, \ldots, \vec{t}_n\right)$, *kde* $\vec{t}_i = (t_{i1}, \ldots, t_{im})$. *Dále mějme atributy* $(A_1, \ldots, A_k)$ *a množinu tříd* $C = (C_1, \ldots, C_l)$, *kde* $k, l, m, n \in \mathbb{N}$

T v rovnici představuje **Rozhodovací strom**, *pro který platí:*

- *každý vnitřní uzel je ohodnocen atributem* A_i
- *každá hrana je ohodnocena predikátem použitelným na atribut rodičovského uzlu* t_{ij}
- *každý list je ohodnocen třídou* C_j

[1]

Výše uvedená definice je vysvětlena na Příkladu 5.1 v Kapitole 5.1. K tomuto příkladu náleží rovněž grafické zobrazení rozhodovacího stromu v podobě Obrázku 10.

2.1.1 Sestavení rozhodovacího stromu

Problém nalezení rozhodovacího stromu se může zdát složité, ale ve skutečnosti se jedná o triviální algoritmus. Jednoduše může být vybrán jeden libovolný atribut a podle něj rozdělit množinu na několik podmnožin a následně tento krok opakovat do té doby, dokud podmnožiny nebudou podle atributů dále dělitelné.

Cílem je nalézt optimální velikost stromu. Malý strom přináší riziko, že nedokáže zahrnout všechny potřebné parametry. Naopak příliš rozsáhlý strom hrozí velkou časovou a prostorovou náročností společně s tím rizikem, že pozbude svou obecnou platnost. Na velké stromy je možné následně aplikovat metodu prořezávání, ale při vytváření rozsáhlejších stromů je jistě jednodušší aplikovat tyto metody již při indukci rozhodovacího stromu.

Pseudokód pro sestavení rozhodovacího stromu je uveden je uveden v Algoritmu 1.

```
1  function DECISION−TREE−LEARNING(example, attributes, default) returns a decision tree
2      inputs: examples, set of examples
3              attributes , set of attributes
4              default, default value for the goal predicate
5   if examples is empty then return default
6   else if all examples have the same classification then return the classification
7   else if attributes is empty then return MAJORITY−VALUE(examples)
8   else
9      best <−− CHOOSE− ATTRIBUTE(attributes, examples)
10     tree <−− a new decision tree with root test best
11     for each value v_i of best do
12        examples_i − {elements of examples with best = v_i}
13        subtree − DECISION−TREE−LEARNING(examples_i, attributes − best, MAJORITY−VALUE(examples)}
14        add a branch to tree with label v_i and subtree subtree
15     end
16  return tree
```

Algoritmus .1: Sestavení rozhodovacího stromu [2]

Sestavením rozhodovacího stromu se zabývá hned několik metod. Mezi nejznámější patří metody TDIDT, ID3, C4.5, ... Těmto metodám je věnována celá Kapitola 5 – Algoritmy využívané při sestavování rozhodovacího stromu.

2.2 Náhodný les

V diplomové práci je použito více rozšířené anglické označení této techniky Random forest, případně pod jeho zkratkou RF.

Random forest je pojem obecné techniky pro učení klasifikace, regrese a dalších úkonů. V základní rovině dále rozpracovává a zdokonaluje teorii rozhodovacích stromů. Proces spočívává v tom, že není sestaven pouze jeden rozhodovací strom (”boosting”), ale je sestavena celá řada rozhodovacích stromů (les). Každý jednotlivý strom zařazený do rozhodovacího procesu pracuje nezávisle na ostatních a má jinou strukturu atributů. Při trénování je sestavena množina atributů. Každý strom je pak následně sestaven na základě podmnožiny, která vznikla náhodným výběrem z těchto atributů sestavených při trénování (bagging). Proběhne test a pokud test označí jeden strom jako pozitivní a další stromy zařazené do rozhodovacího procesu jako negativní, je pravděpodobné, že strom č. 1 udělal při klasifikaci chybu.

Výhoda Random forestu spočívá především v eliminaci chyb při rozhodování jednotlivého stromu. Tato schopnost eliminace chyb má své využití především ve sporných případech. Další výhodou RF je ta vlastnost, že metoda dokáže sama klasifikovat důležitost jednotlivých atributů.

Sestavením Random forestu se podrobně zabývá Kapitola 6.1 – Vytvoření Random forestu.

2.3 Support Vector Machine

Support vector machine (SVM) je metoda klasifikace lineárních dat. Při klasifikaci nelineárních dat musí být data transformována do prostoru s vyšší dimenzí, transformační funkci nazýváme kernel. Následně je možné tato data klasifikovat jako lineární. Tato metoda se často využívá v aplikacích pro vyhledávání objektu v obrazech. Samotné trénování je časově a paměťově náročné.

Podstatou trénování s SVM je vytvoření hyperroviny (prostor $n-1$ dimenze), která ze vstupních dat vytvoří 2 sady vektorů v n-rozměrném prostoru. Pro každou sadu se potom vytvoří takováto hyperrovina, která se znovu použije jako vstupní sada. Data, která není možné separovat lineárně mohou být nejprve transformována do prostoru s vyšší dimenzí.

Pro všechna x_i, x_j z prostoru $\mathcal{X}$, můžeme jisté funkce $k(x_i, x_j)$ vyjádřit jako skalární součin v prostoru $\mathcal{V}$. Funkci $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ pak nazýváme kernel. V případě strojového učení si situaci zjednodušíme pomocí takzvané ”feature mapy”$\varphi : \mathcal{X} \to \mathcal{V}$. Kernel poté můžeme zapsat jako

$$K(x_i, x_j) = \left\langle \varphi(x_i), \varphi(x_j) \right\rangle_{\mathcal{V}}, \tag{1}$$

kde $\langle \cdot, \cdot \rangle_{\mathcal{V}}$ a $\mathcal{V}$ je prostor se skalárním součinem. [7]

V praktické části budeme pracovat se dvěma typy kernelů. Prvním zástupcem bude Lineární kernel. Lineární kernel spočítáme

$$K(x_i, x_j) = x_i^T x_j, \tag{2}$$

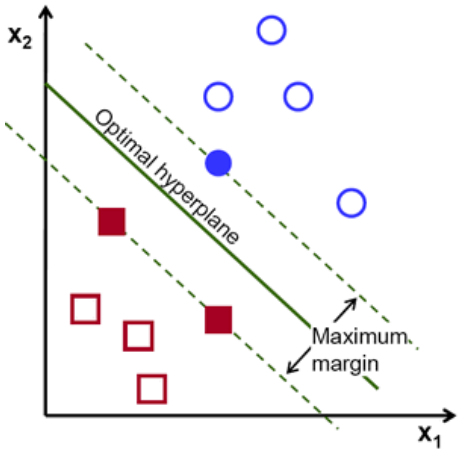
kde $x_i, x_j \in \mathcal{X}$.

Druhým zástupcem je kernel RBF (Radial basis function). Kernel je vyjádřen funkcí

$$K(x_i, x_j) = e^{-\gamma \|x_i - x_j\|^2}, \tag{3}$$

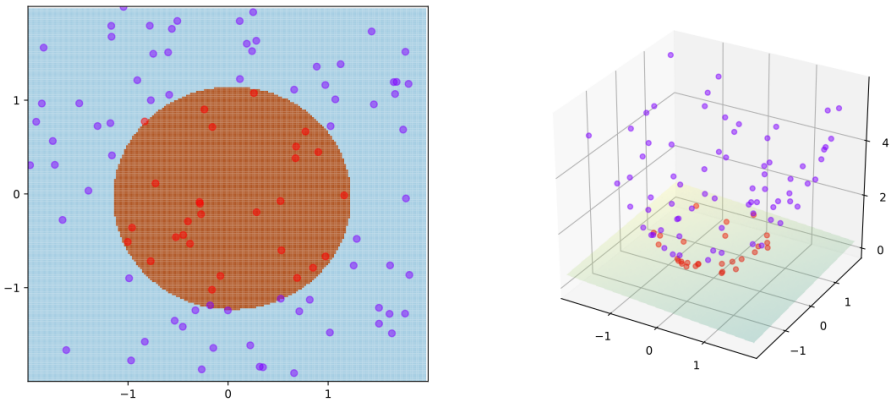
kde $x_i, x_j \in \mathcal{X} \in \mathbb{N}$ a $\gamma \in (0, +\infty)$.

Pro lepší pochopení lineárního kernelu se podívejme na Obrázek 1. Zřejmě platí $x_1 \in \mathbb{R}, x_2 \in \mathbb{R}$, takže $\mathcal{V} \in \mathbb{R}^2$. Podle definice bude tedy hyperrovina v dimenzi $n - 1 = 1$, tedy přímka. Rozdělení prostoru je tedy min-max úloha, kterou zjednodušeně můžeme popsat: Najdi hyperrovinu f, jejíž minimální vzdálenost (v našem případě kolmice) od krajních bodů dvou skupin bodů bude maximální.



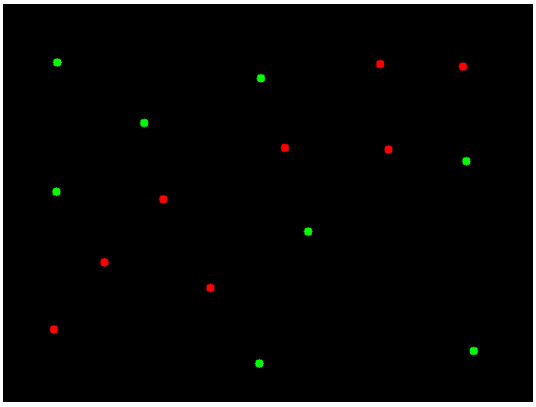
Obrázek 1: Příklad optimálního rozdělení bodu v 2D [3]

Na Obrázku 2 je znázorněna situace, kdy klasifikovaná data nelze ve dvou dimenzích lineárně oddělit. Pokud, ale data transformujeme do třetí dimenze například pomocí feature mapy $\varphi(a, b) = (a, b, a^2+b^2)$, tedy kernelu $K(x, y) = xy+x^2y^2$ je nalezení hyperroviny $n - 1 = 2$, tedy plochy, opět jednoduchý min-max úkol.

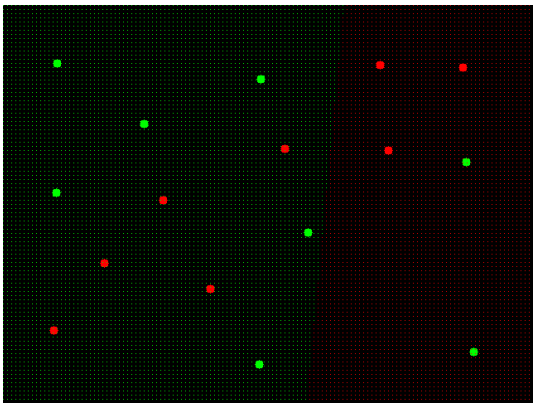


Obrázek 2: Příklad optimálního rozdělení bodu v 3D [7]

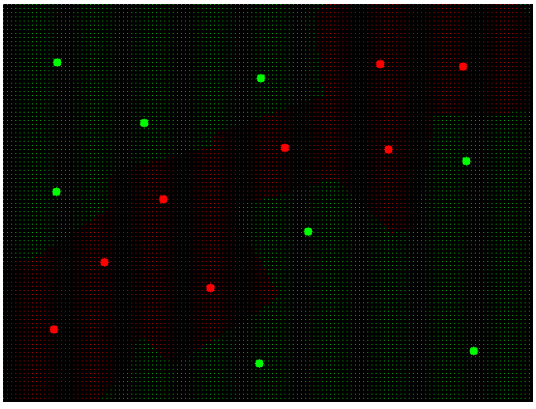
Tento odstavec bude věnován grafickému oddělení skupiny bodů pomocí metody SVM Linear a SVM RBF. Obrázek 3a znázorňuje trénovací množinu se kterou budou pracovat výše uvedené metody SVM. Platí, že dané body v prostoru $\mathbb{R}^2$ nelze od sebe spolehlivě lineárně oddělit. Obrázek 3b demonstruje rozdělení trénovací množiny pomocí metody SVM Linear. Šrafovaná oblast zobrazuje předpověď SVM modelu pro další body. Nachází -li se červený bod v zelené oblasti nebo naopak, jedná se o chybu v klasifikaci bodu, Kapitola 2.6 – Hodnocení binárních klasifikátorů. Pokud, ale vstupní data převedeme do prostoru vyšší dimenze například použitím kernelu RBF bude výsledek uvedený na Obrázku 3c přijatelnější. Velikost oblastí okolí červených bodů je možné upravit pomocí parametru γ.



(a) Trénovací množina



(b) SVM Linear



(c) SVM RBF

Obrázek 3: Srovnání SVM Linear a SVM RBF

2.4 Entropie

Entropie je jedním ze základních pojmů ve fyzice, matematice, teorii pravděpodobnosti, teorii informace a v mnoha dalších oblastech vědy. Setkat se s ní můžeme všude tam, kde hovoříme o pravděpodobnosti možných stavů dané soustavy nebo systému.

Velmi zjednodušeně by se dalo říci, že entropie je míra neuspořádanosti stavů v soustavě.

V této práci se budeme setkávat především s entropií diskrétních stavů zpracovanou Joasiahem Willardem Gibbsem.

Definice 2.2 *Nechť S je funkcionál nad diskrétní pravděpodobnostní funkcí P, $k \in \mathbb{R}$ je konstanta jednotek ve kterých entropii S měříme a P_i je pravděpodobnost i-tého mikrostavu. Pak*

$$S = -k \sum_i P_i \ln P_i. \tag{4}$$

Dále pracujeme s jednotkami bitů, pro které platí $k = \frac{1}{\ln(2)}$. Abychom odlišili tento konkrétní případ entropie, bude v další části diplomové práce použito označení entropie H. Po matematické úpravě získáme výsledný stav entropie ve tvaru

$$H = -k \sum_i P_i \ln P_i = -\frac{1}{\ln 2} \sum_i P_i \ln P_i = -\sum_i P_i \frac{\log_2 e}{\log_2 2} \cdot \frac{\log_2 P_i}{\log_2 e} = -\sum_i P_i \log_2 P_i \tag{5}$$

2.5 Klasifikační a regresní problém

Klasifikační problém je pojem používaný pro nalezení zobrazení $f : D \to C$, kde je ke každému prvku t_i z množiny $D = \{t_1, \ldots, t_n\}$ přiřazen právě jeden prvek c_j z množiny $C = \{c_1, \ldots, c_m\}$. Proces klasifikace se skládá ze dvou kroků:

1. *učení, trénování*: představuje tvorbu klasifikačního modelu pomocí trénovací, předem dané, množiny,
2. *vlastní klasifikace*: použití klasifikačního modelu pro určení příslušné třídy pro testovací data.

Regresní problém je hledání řešení pro nalezení regresního modelu $Y_i = \alpha + \beta x_i + \epsilon_i$, pro $i = 1, 2, \ldots$, kde chyba ϵ_i nemá symetrické rozložení.

2.6 Hodnocení binárních klasifikátorů

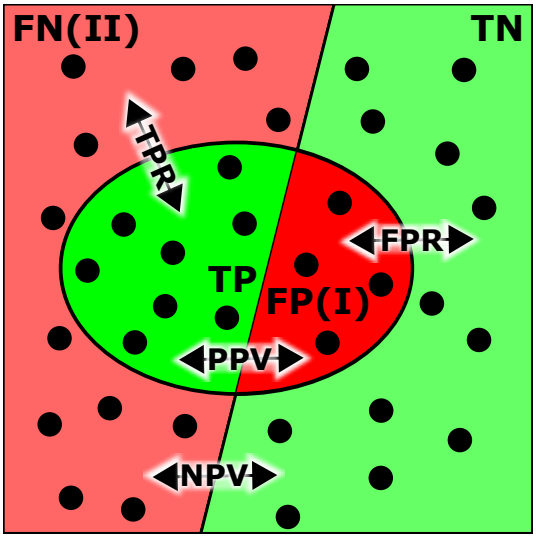
Aby bylo možné zkoumat a porovnávat klasifikátory zavádíme 4 základní stavy ve kterých se může klasifikátor ocitnout.

- TP – True positive, správně vyhodnocený pozitivní výskyt
- TN – True negative, správně vyhodnocený negativní výskyt
- FP – False positive, označujeme také jako chybu I. typu. Význam této hodnoty je, že výskyt byl vyhodnocen jako pozitivní, ale ve skutečnosti se jedná o negativní výskyt
- FN – False negative, označujeme jako chybu II. typu. Výskyt byl vyhodnocen klasifikátorem jako negativní, ale ve skutečnosti se jedná o pozitivní výskyt

Pro lepší přehlednost a nezávislost na velikosti testovací množiny zavádíme poměrové hodnocení klasifikátoru.

- $\mathrm{ACC} = \frac{\mathrm{TP}+\mathrm{TN}}{\mathrm{TP}+\mathrm{TN}+\mathrm{FP}+\mathrm{FN}}$ – Přesnost. Vyjadřuje poměr mezi správně vyhodnocenými výskyty a všemi výskyty
- $\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}}$ – True positive rate. Vyjadřuje poměr mezi správně vyhodnocenými pozitivními výskyty a všemi pozitivními výskyty
- $\mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{TN}+\mathrm{FP}}$ – False positive rate. Vyjadřuje poměr mezi chybně vyhodnocenými negativními výskyty a všemi negativními výskyty
- $\mathrm{PPV} = \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FP}}$ – Positive predictive value. Popisuje kvalitu diagnostického testu nebo statistického měření vzhledem k pozitivní předpovědi. Jedná se o poměr mezi správně vyhodnocenými pozitivními výskyty a součtem správně vyhodnocených pozitivních výskytů a chybně vyhodnocených negativních výsledků.
- $\mathrm{NPV} = \frac{\mathrm{TN}}{\mathrm{TN}+\mathrm{FN}}$ – Negative predictive value. Popisuje kvalitu diagnostického testu nebo statistického měření vzhledem k negativní předpovědi. Jedná se o poměr mezi správně vyhodnocenými negativními výskyty a součtem správně vyhodnocených negativních výskytů a chybně vyhodnocených pozitivních výsledků.
- $\mathrm{F}_1 = 2 \cdot \frac{\mathrm{PPV} \cdot \mathrm{TPR}}{\mathrm{PPV}+\mathrm{TPR}}$ – F_1 Score se používá k testu přesnosti (ve smyslu správnosti klasifikace) a to především v případech, kdy je velké množství negativních záznamů, které nejsou pro test přesnosti relevantní. Jedná se o harmonický průměr přesnosti (PPV) a citlivosti (TPR).

Na Obrázku 4 jsou znázorněny vazby mezi některými výše popsanými stavy.



Obrázek 4: Základní metriky hodnocení klasifikátorů [6]

3 Popis vlastností obrazové funkce

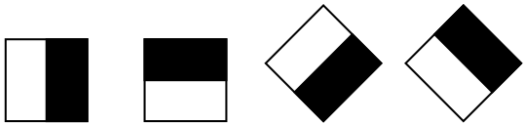
V této kapitole jsou popsány tři základní metody běžně využívané v praxi. Každá z těchto metod charakterizuje obrazovou funkci jiným způsobem. V podkapitolách je vysvětlen základní princip.

3.1 Haarovy příznaky

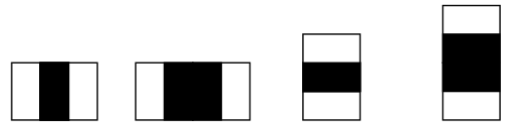
Haarovy příznaky svým charakterem detekují specifické rysy obrazu, které jsou založeny na jasových složkách digitálního obrazu. Jedná se o jednoduché obdélníkové oblasti (Obrázek 5) a můžeme je rozdělit do několika typů podle informace, která má být detekována. Například na hranové (5a), čárové (5b) a středové (5c) příznaky. Hodnota příznaku se vypočítává z intenzity obrazu pod danou oblastí.

$$f(x) = w_0 r_0 + w_1 r_1, \tag{6}$$

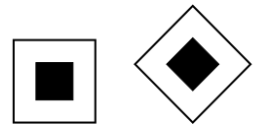
kde $f(x)$ je odezva Haarova příznaku na snímek x, w_0 je váha bílé obdélníkové oblasti r_0 a w_1 je váha černé obdélníkové oblasti r_1. [10]



(a) Hranové příznaky



(b) Čárové příznaky

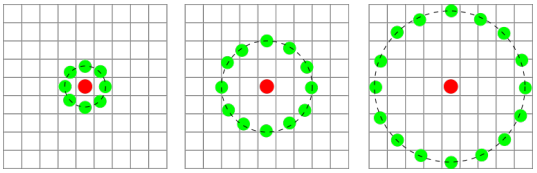


(c) Středové příznaky

Obrázek 5: Haarovy příznaky

3.2 Lokální binární vzor

Metoda lokálních binárních vzorů (Local Binary Pattern - LBP) slouží stejně jako Haarovy příznaky k popisu vlastností obrazu. Popis textury se provádí v blízkém okolí jednotlivých pixelů vstupního obrazu. Pro okolí byla zvolena kruhová reprezentace a výsledný popis se tedy vztahuje k bodu ležícímu ve středu okolí. Viz Obrázek 6



Obrázek 6: Příklady okolí pro výpočet LBP

Texturu T v lokálním okolí bodu definujeme jako:

$$T = t\left(g_c, g_0, \cdots, g_{P-1}\right), \tag{7}$$

kde g_c je hodnota pixelu ve středu lokálního okolí a g_p, kde $p = 0, \cdots, P-1$ jsou hodnoty pixelů $P > 1$ symetricky rozmístěných na kružnici o poloměru $R > 0$ se středem ve zkoumaném bodě. [11] Tedy

$$t = \sum_{p=0}^{P-1} g_p - g_c. \tag{8}$$

Od deskriptoru předpokládáme odolnost vůči jasovým změnám obrazu. Abychom této odolnosti dosáhli upravíme vzorec pro výpočet a LBP tedy definujeme:

$$\begin{aligned} LBP_{P,R} &= \sum_{p=0}^{P-1} s\left(g_p - g_c\right) 2^p \\ s(x) &= \begin{cases} 1 & \text{pro } x \geq 0 \\ 0 & \text{pro } x < 0 \end{cases} \end{aligned} \tag{9}$$

Pro uniformní a rotačně invariantní vzory použijeme vzorec

$$\begin{aligned} LBP_{P,R} &= \begin{cases} \sum_{p=0}^{P-1} s\left(g_p - g_c\right) 2^p & \text{pro } U\left(LBP_{P,R}\right) \leq 2 \\ P+1 & \text{pro } U\left(LBP_{P,R}\right) > 2 \end{cases} \\ U\left(LBP_{P,R}\right) &= \left|s\left(g_{P-1} - g_c\right) - s\left(g_0 - g_c\right)\right| + \sum_{p=1}^{P-1} \left|s\left(g_p - g_c\right) - s\left(g_{p-1} - g_c\right)\right| \end{aligned} \tag{10}$$

3.3 HOG

Metoda HOG (Histograms of Orinted Gradiets) byla vyvinuta za účelam detekce lidských postav v obraze. Příznaky jsou založeny na hledání významných hran v obraze. Jednotlivé gradienty jsou reprezentovány svou velikostí a směrem. Jednotlivé gradienty jsou získány konvolucí Gaussovsky filtrovaného obrazu I s maskou $[-1, 0, 1]$, resp $[-1, 0, 1]^T$

$$\begin{aligned} I_x &= I * [-1, 0, 1], \\ I_y &= I * [-1, 0, 1]^T, \end{aligned} \tag{11}$$

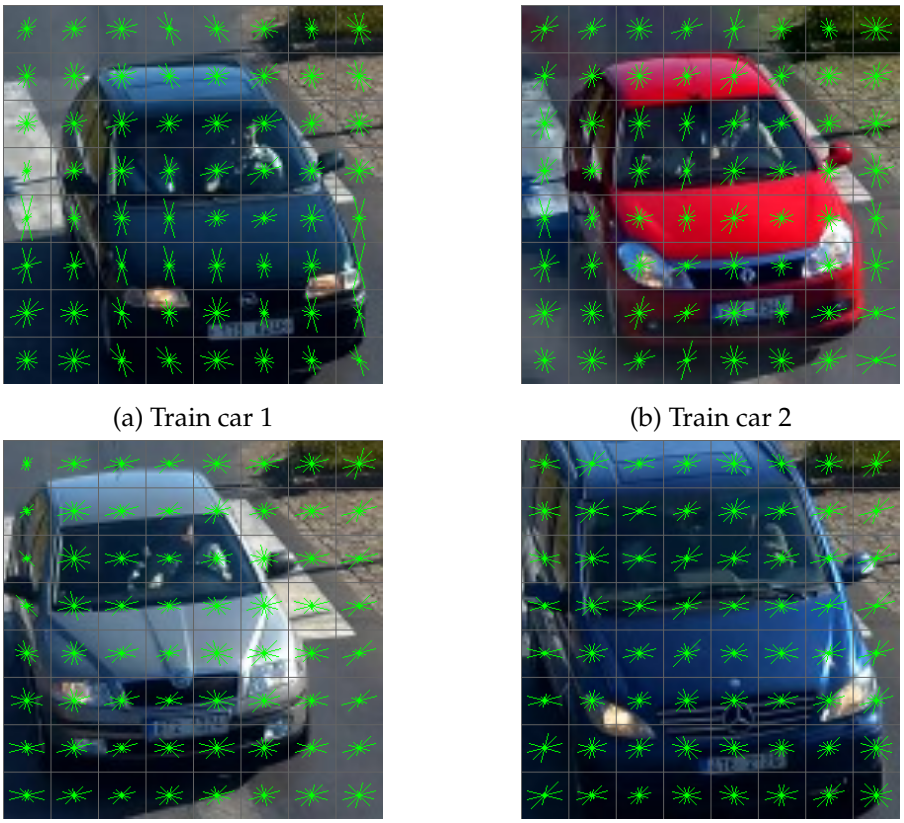
kde * značí konvoluci.

Poté, co jsou získány obrazy I_x a I_y, je pro každou buňku vypočtena velikost gradientu $m(x, y)$ a směr $\Theta(x, y)$.

$$\begin{aligned} m(x, y) &= \sqrt{I_x^2 + I_y^2}, \\ \Theta(x, y) &= \tan^{-1}\left(\frac{I_y}{I_x}\right). \end{aligned} \tag{12}$$

Z vypočtených hodnot je sestaven histogram orientací. Oblast $i = \{0, 2\pi\}$ rozdělíme pod několika stejně velkých výsečí (binů) a zaznamenáme přínos jednotlivých gradientů. [11]

Konstrukce HOG deskriptoru spočívá v rovnoměrném rozdělení vstupního obrazu do stejně velkých čtvercových bloků. Bloky následně rozdělíme do buněk, ve kterých počítáme gradienty. Výsledný obraz tedy můžeme znázornit jak je uvedeno na Obrázku 7. Zde je přínos jednotlivých gradientů rozdělen do šesti binů.



(a) Train car 1

(b) Train car 2

(c) Train car 3

(d) Train car 4

Obrázek 7: Znázornění výpočtu přínosu jednotlivých gradienů

4 Vhodné problémy pro řešení rozhodovacím stromem

Jak již bylo řečeno výše v Kapitole 2.1, smyslem rozhodovacího stromu je zařazení vstupního objektu do jedné z výsledných tříd. Optimálních výsledků bude dosaženo, pokud rozhodovací strom bude řešit situace s následujícími charakteristikami.

- Instance jsou reprezentovány atributy s konkrétní hodnotou:
  - konečný počet atributů (např. barva auta, . . .) a každá instance má právě jednu hodnotu tohoto atributu (např. červená, zelená, . . .),
  - když má každý atribut malý počet diskrétních hodnot (červená, zelená, modrá) je pro rozhodovací strom jednodušší nalézt řešení,
  - algoritmus může být rozšířen aby zvládal také reálné hodnoty (např. teplota okolí, úhrn srážek, . . .).
- Rozhodovací funkce v uzlu má diskrétní počet výstupů:
  - rozhodovací strom klasifikuje každou hodnotu jako jeden z výstupů. Nejjednodušší je případ, kdy jsou možné právě dvě hodnoty (booleovská klasifikace).
  - je možné, aby cílová funkce měla reálný výstup, ale tato varianta se běžně nepoužívá.
- Disjunktní rozdělení hodnot.
  - Rozhodovací stromy přirozeně reprezentují disjunktní množiny.
- Trénovací data mohou obsahovat malé procento chyb.
  - Na rozdíl od jiných metod je výhoda rozhodovacího stromu v tom, že trénovací množina může obsahovat malé procento chyb.
- Trénovací data mohou obsahovat chybějící atributy.
  - Metody pro rozhodovací stromy mohou být použity i pro data s chybějícími atributy.

Protože v praxi nastane jen málo jevů, kdy jsou splněny všechny tyto podmínky, je složité pomocí jednoho rozhodovacího stromu dojít k bezchybné klasifikaci. K eliminaci této negativní situace tedy můžeme použít mimo jiné metodu Random forest. Ta, jak již bylo řečeno v Kapitole 2.2, sestaví řadu rozhodovacích stromů. Následně je zde uplatněn většinový princip.

5 Algoritmy využívané při sestavování rozhodovacího stromu

Tato kapitola je věnována algoritmům, pomocí kterých se dá rozhodovací strom sestavit. Jako ukázkový příklad si jsem zvolil sestavení rozhodovacího stromu pomocí algoritmu TDIDT.

5.1 Algoritmus TDIDT

TDIDT (Top Down Induction of Decision Tree) funguje na principu sestavení stromu od shora dolů. V zásadě vybere jeden atribut jako kořen dílčího stromu. Následně rozdělí data v tomto uzlu na podmnožiny podle hodnot atributů. V dalším kroku přidá uzel pro každou podmožinu. Existuje -li uzel, ve kterém je více než jedna třída, opakuje běh pro tento uzel.

V algoritmu TDIDT se využívá tak zvané Occamovy břitvy, která říká: *Entity se nemají zmnožovat více, než je nutné*. V tomto příkladě ji budeme interpretovat následovně: Čím má parametr vyšší vypovídací hodnotu (menší entropii 2.4), tím je lepším kandidátem na kořen stromu (později podstromu).

Jako nástin algoritmu si představme následující situaci.

Příklad 5.1
Majitel řetězce restaurací chce předvídat chování zákazníků. Přesněji se snaží zjistit, jestli zákazník počká na uvolnění stolu nebo ne. Proto provádí po nějaký čas pozorování okolí restaurací a zaměřil se na tyto aspekty:

1. *Alternativa*: v okolí se nachází srovnatelná restaurace
2. *Bar*: restaurace má pohodlné zázemí, kde může zákazník počkat
3. *Pá/So*: den v týdnu (pátek nebo sobota = *T*, jiný den *F*)
4. *Hlad*: zákazník je hladový
5. *Obsazenost*: počet lidí v podniku (hodnoty *Nikdo*, *Někdo*, *Plno*)
6. *Cena*: cenová skupina restaurace ($, $$, $$$)
7. *Dešť*: jestli venku prší
8. *Rezervace*: zákazník udělal rezervaci

Zákazník	Alt	Bar	Pá/So	Hlad	Obs	Cena	Déšť	Rez	Typ	Čas	Počká
X_1	T	F	F	T	*Někdo*	\$\$\$	F	T	*Fr*	$0-10$	T
X_2	T	F	F	T	*Plno*	\$	F	F	*Th*	$30-60$	F
X_3	N	Y	N	N	*Někdo*	\$	F	F	*Bg*	$0-10$	T
X_4	T	F	T	T	*Plno*	\$	F	F	*Th*	$10-30$	T
X_5	T	F	T	F	*Plno*	\$\$\$	F	T	*Fr*	≥ 60	F
X_6	F	T	F	T	*Někdo*	\$\$	T	T	*It*	$0-10$	T
X_7	F	T	F	F	*Nikdo*	\$	T	F	*Bg*	$0-10$	F
X_8	F	F	F	T	*Někdo*	\$\$	T	T	*Th*	$0-10$	T
X_9	F	T	T	F	*Plno*	\$	T	F	*Bg*	≥ 60	F
X_{10}	T	T	T	T	*Plno*	\$\$\$	F	T	*It*	$10-30$	F
X_{11}	F	F	F	F	*Nikdo*	\$	F	F	*Th*	$0-10$	F
X_{12}	T	T	T	T	*Plno*	\$	F	F	*Bg*	$30-60$	T

Tabulka 1: Trénovací množina k příkladu 5.1

9. *Typ*: typ restaurace (Francouzská (*Fr*), Italská (*It*), Thajská (*Th*), Burger (*Bg*))
10. *Čas*: očekávaná doba čekání na místo ($0-10$ minut, $10-30$, $30-60$, ≥ 60)

Majitel řetězce nashromáždil údaje do Tabulky 1. Tuto tabulku nazýváme trénovací množina.

Kořen stromu zvolíme podle nejnižší entropie. Tedy vypočítáme entropii podle rovnice 5 pro jednotlivé stavy. Jako úspěch (p_+) zvolíme, že zákazník počká jako neúspěch (p_-), že zákazník nepočká.

$$\begin{aligned} H(Obsazenost) &= \frac{2}{12}H(Obsazenost(Nikdo)) + \frac{4}{12}H(Obsazenost(Nekdo)) + \frac{6}{12}H(Obsazenost(Plno)) \\ &\approx \frac{2}{12}\cdot 0 + \frac{4}{12}\cdot 0 + \frac{6}{12}\cdot 0.918 \\ &\approx 0.459 \end{aligned} \tag{13}$$

Dosazením výsledků rovnic 14, 15 a 16 do rovnice 13 získáme výsledek $H(Obsazenost) \approx= 0.459$

Parametr	H(Parametr)
Obsazenost	0.459
Čas	0.459
Cena	0.804
Hlad	0.804
Rezervace	0.879
Pá/So	0.879
Bar	0.981
Typ	0
Alternativa	0
Déšť	0

Tabulka 2: Entropie pro jednotlivé parametry příkladu 5.1

Entropie v případě, že nikdo nesedí v restauraci

$$\begin{aligned} H(Obsazenost(Nikdo)) &= -p_+ \log_2 p_+ - p_- \log_2 p_- \\ &= -\left(\frac{0}{2}\right)\log_2\left(\frac{0}{2}\right) - \left(\frac{2}{2}\right)\log_2\left(\frac{2}{2}\right) \\ &= 0 \end{aligned} \tag{14}$$

Entropie v případě, že je několik míst v restauraci obsazeno

$$\begin{aligned} H(Obsazenost(Nekdo)) &= -p_+ \log_2 p_+ - p_- \log_2 p_- \\ &= -\left(\frac{4}{4}\right)\log_2\left(\frac{4}{4}\right) - \left(\frac{0}{4}\right)\log_2\left(\frac{0}{4}\right) \\ &= 0 \end{aligned} \tag{15}$$

Entropie v případě, že je restaurace plná

$$\begin{aligned} H(Obsazenost(Plno)) &= -p_+ \log_2 p_+ - p_- \log_2 p_- \\ &= -\left(\frac{2}{6}\right)\log_2\left(\frac{2}{6}\right) - \left(\frac{4}{6}\right)\log_2\left(\frac{4}{6}\right) \\ &\approx 0.918 \end{aligned} \tag{16}$$

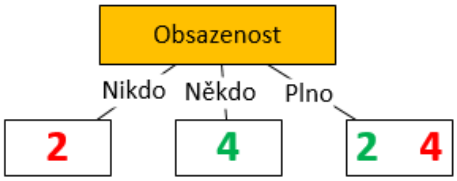
Podobně pokračujeme s výpočtem entropií pro ostatní atributy. Jednotlivé výsledky jsou uvedeny v Tabulce 2.

V tomto případě lze tedy zvolit parametr Obsazenost nebo Čas, které poskytují největší informační zisk o rozdělení množiny do podmnožin na základě jednoho parametru,

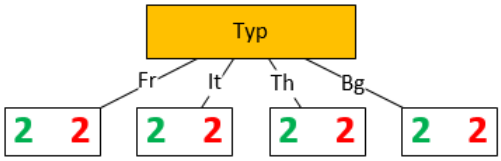
znázorněno na Obrázku 8. Dále by algoritmus pokračoval výpočtem dalších dílčích podstromů, dokud by nezískal kompletní informaci o rozdělení.

V prvním kroku nemají parametry Typ restaurace, Alternativa nebo Déšť žádnou vypovídající hodnotu viz. Obrázek 9.

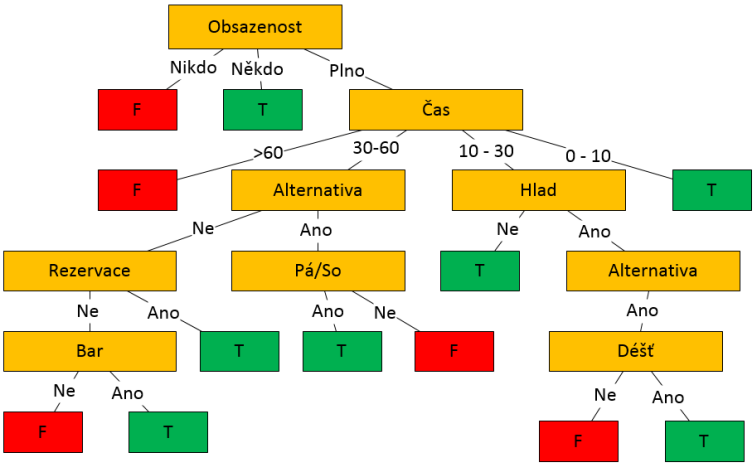
Výsledný rozhodovací strom by mohl mít podobnou strukturu jak je uvedeno na Obrázku 10.



Obrázek 8: Rozdělení množiny podle obsazenosti



Obrázek 9: Rozdělení množiny podle typu



Obrázek 10: Možný tvar stromu

Více informací k tomuto příkladu uvedeno v knize [2] ■

5.2 Algoritmus ID3

Algoritmus ID3 byl vyvinut Rossem Quinlanem už v roce 1986. Tento algoritmus využívá k učení funkce s booleovskými proměnnými. Podobně jako algoritmus TDIDT 5.1 vytváří strom od shora dolů. V každém uzlu zvolí atribut, který nejlépe klasifikuje lokální trénovací data (nejlepší atribut má nejvyšší informační zisk). Takto algoritmus pokračuje, dokud nejsou správně zařazena všechna trénovací data, a nebo dokud nebyly využity všechny atributy.

```
1  ID3 (Examples, Target_Attribute, Attributes)
2      Create a root node for the tree
3      If all examples are positive, Return the single−node tree Root, with label = +.
4      If all examples are negative, Return the single−node tree Root, with label = −.
5      If number of predicting attributes is empty, then Return the single node tree Root,
6      with label = most common value of the target attribute in the examples.
7      Otherwise Begin
8          A <−− The Attribute that best classifies examples.
9          Decision Tree attribute for Root = A.
10         For each possible value, vi, of A,
11             Add a new tree branch below Root, corresponding to the test A = vi.
12             Let Examples(vi) be the subset of examples that have the value vi for A
13             If Examples(vi) is empty
14                 Then below this new branch add a leaf node with label = most common target value in the
                       examples
15             Else below this new branch add the subtree ID3 (Examples(vi), Target_Attribute, Attributes − A )
16     End
17     Return Root
```

Algoritmus .1: Algoritmus ID3 [5]

5.3 Algoritmus C4.5 a C5.0

Algoritmus C4.5 má oproti TDIDT a ID3 několik zásadních vylepšení. Zejména se jedná o to, že je strom prořezaný již při jeho konstrukci. Navíc Algoritmus C4.5 umožňuje práci s numerickými atributy, chybějícími hodnotami a také brát do úvahy různé ceny za různá chybná rozhodnutí. Můžeme tedy prioritizovat jistou hodnotu oproti jiné. Tato vlastnost je velmi výhodná v případě, že očekáváme pravděpodobnost určitého jevu jako velmi malou nebo naopak velkou. Rozhodovací strom pak může být výrazně jednodušší a tedy i rozhodování je rychlejší.

Algoritmus C5.0 (resp. See5) implementuje oproti C4.5 několik nových funkcí. Za zmínku rozhodně stojí parametr cena za nesprávnou klasifikaci (v algoritmus C4.5 jsou všechny chyby považovány za sobě rovné, to ovšem v reálném světě vždy neplatí). Dále je algoritmus C5.0 schopen pracovat s více datovými typy, například časem.

6 Sestavení Random forestu

Algoritmus Random forestu (viz. Kapitola 2.2) byl vyvinut Leem Breimanem a Adele Cutlerovou. Tento algoritmus si dokáže poradit jak s klasifikačním, tak regresním problémem (viz. Kapitola 2.5). OpenCV na Random forest pohlíží jako na matici rozhodovacích stromů. Klasifikace funguje následovně: Random forest přijme vektor vstupních vlastností, klasifikuje jej pomocí všech stromů v lese a vrátí třídu, která byla vyhodnocena jako nejpravděpodobnější.

Všechny stromy se učí za použití stejných parametrů, ale na různých trénovacích množinách. Tyto podmnožiny jsou generovány z originální trénovací množiny pomocí samozaváděcí procedury: pro každou trénovací podmnožinu vybere náhodně stejný počet trénovacích vektorů jako je v celé trénovací množině. Algoritmus připouští opakování vektorů, takže v jednotlivých stromech budou některé vektory použity vícekrát a některé budou chybět. Žádný z rozhodovacích stromů není prořezaný.[3]

$$S = \begin{pmatrix} f_{11} & \cdots & f_{m1} & C_1 \\ & \ddots & & \vdots \\ \vdots & f_{AB} & \vdots & C_B \\ & & \ddots & \vdots \\ f_{1n} & \cdots & f_{mn} & C_i \end{pmatrix}, \tag{17}$$

kde f_{AB} je vlastnost A trénovacího objektu B, který kategorizujeme jako class C_B.

6.1 Vytvoření Random forestu

Vytvoření rozhodovacího stromu spočívá v náhodném výběru k řádků, kde $k \in \mathbb{N}$ a zároveň $k \leq n$ z trénovací množiny uvedené v úvodu této kapitoly a následném sestavení rozhodovacího stromu z těchto dat. Random forest je potom $l \in \mathbb{N}$ rozhodovacích stromů.

Příklad 6.1
Mějme trénovací množinu S z úvodu Kapitoly 17. Nyní vytvořme l náhodných podmnožin $S_1, S_2, \ldots, S_l$.

$$S_1 = \begin{pmatrix} f_{12} & \cdots & f_{m2} & C_2 \\ f_{15} & \cdots & f_{m5} & C_5 \\ f_{18} & \cdots & f_{m8} & C_8 \end{pmatrix}, \quad S_2 = \begin{pmatrix} f_{13} & \cdots & f_{m3} & C_3 \\ f_{15} & \cdots & f_{m5} & C_5 \\ f_{17} & \cdots & f_{m7} & C_7 \\ f_{18} & \cdots & f_{m8} & C_8 \\ f_{19} & \cdots & f_{m9} & C_9 \end{pmatrix}, \tag{18}$$

$$S_l = \begin{pmatrix} f_{1a} & \cdots & f_{ma} & C_a \\ \vdots & \ddots & \vdots & \vdots \\ f_{1b} & \cdots & f_{mb} & C_b \end{pmatrix},$$

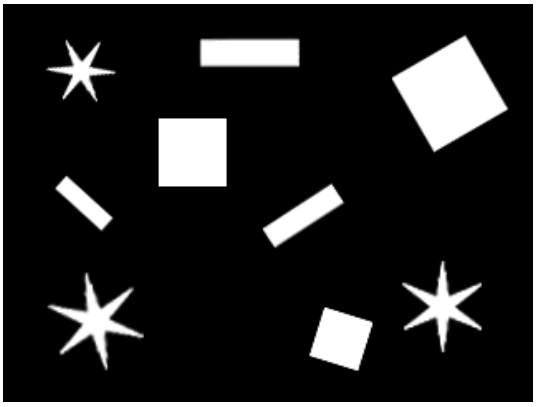
kde $a, b, m, l, i \in \mathbb{N}$, f_{mn} je vlastnost objektu, C_i je výsledná klasifikační třída.

Následně se nad takto vytvořenými maticemi spustí algoritmus pro vytvoření rozhodovacího stromu (například TTDI popsaný v předchozí Kapitole 5.1).

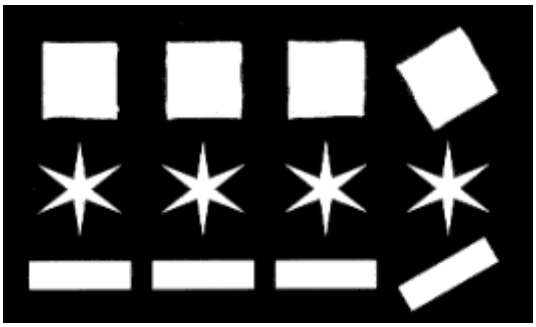
Předpokládejme, že jsme podle popsaného postupu sestavili l rozhodovacích stromů. Tuto množinu již nazýváme Random forest. Každý z těchto stromů je naučený podle jiných parametrů, takže může testovaný objekt zařadit do jiné třídy. Pošleme -li do tohoto lesa testovací objekt, obdržíme výsledek následujícího typu: pravděpodobnost výskytu tohoto objektu ve třídě C_i je k %. Dále je už rozhodnutí jen na vhodném použití parametrů, kdy je pravděpodobnost dostatečně velká, abychom tento objekt označili, že skutečně patří do dané třídy. ■

V tuto chvíli na okamžik opustíme obecnou rovinu a zkusíme jednoduchý praktický příklad.

Příklad 6.2
Mějme jednoduchou úlohu rozpoznání objektů v obraze. Našim úkolem bude vytvořit Random forest pro rozpoznání čtverce, hvězdičky a obdélníku v testovacím Obrázku 11. Aby bylo možné tento úkol splnit potřebujeme připravit také trénovací množinu z Obrázku 12.



Obrázek 11: Testovací obrázek



Obrázek 12: Trénovací obrázek

Nejprve je potřeba zvolit vhodné parametry jak kvalifikovat jednotlivé objekty. Na rozpoznání hvězdičky bude nejlepší použít obvod (hvězdička má jistě větší obvod než čtverec, či obdélník). Tento parametr nám však již nebude stačit pro rozlišení obdélníku a čtverce. Zde se nabízí použití rozptylu délek hran objektu. Pro rozpoznání je potřeba zvolit prarametry tak, aby byly invariantní vůči poloze a rotaci a velikosti objektu.

K výpočtu bude potřeba ještě několik vzorců. Vzorec pro centrální moment, těžiště a moment invariantní vůči poloze objektu.

Těžiště objektu spočítáme pomocí vzorce

$$x_T = \frac{m_{1,0}}{m_{0,0}}, \quad y_T = \frac{m_{0,1}}{m_{0,0}}, \tag{19}$$

kde $m_{i,j}$ je obecný moment.

Definice 6.1 *Obecný moment definujeme*

$$\begin{gathered} m_{p,q} = \int\int x^p y^q f(x,y)\,\mathrm{d}x\mathrm{d}y, \\ \text{diskrétně} \\ m_{p,q} = \sum\sum x^p y^q f(x,y), \end{gathered} \tag{20}$$

kde $p, q \in \mathbb{N}$ řád momentu. Integrujeme, případně sčítáme přes všechny body objektu.

V našem případě, ale potřebujeme moment, který nebude závislý na poloze. Tuto vlastnost splňuje centrální moment.

Definice 6.2 *Centrální moment definujeme*

$$\begin{gathered} \mu_{p,q} = \int\int (x-x_T)^p (y-y_T)^q f(x,y)\,\mathrm{d}x\mathrm{d}y, \\ \text{diskrétně} \\ \mu_{p,q} = \sum\sum (x-x_T)^p (y-y_T)^q f(x,y), \end{gathered} \tag{21}$$

kde x_T a y_T je poloha těžiště objektu a $p, q \in \mathbb{N}$ řád centrálního momentu. Integrujeme, případně sčítáme přes všechny body objektu.

Jako první parametr tedy zvolíme poměr obvodu k obsahu obrazce. Ukázalo se, že je vhodné použít $f_1 = \frac{p^2}{\mu_{0,0}}$, kde p je obvod obrazce a $\mu_{0,0}$ je nultý centrální moment obrazce. Zanedbáme -li jasovou funkci $f(x,y)$, rozuměj bude mít hodnotu 1 v každém bodě objektu, bude se jednat o obsah.

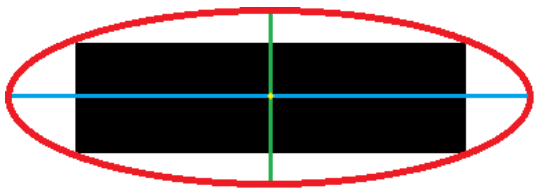
Informaci ohledně orientace obrazu můžeme odvodit pomocí centrálních momentů druhého řádu k sestavení kovarianční matice.

$$\mu'_{20} = \frac{\mu_{20}}{\mu_{00}}, \quad \mu'_{02} = \frac{\mu_{02}}{\mu_{00}}, \quad \mu'_{11} = \frac{\mu_{11}}{\mu_{00}}$$

Kovarianční matici tedy můžeme zapsat ve tvaru

$$\mathrm{cov}\,(I(x,y)) = \begin{pmatrix} \mu'_{20} & \mu'_{11} \\ \mu'_{11} & \mu'_{02} \end{pmatrix} \tag{22}$$

Vlastní čísla kovarianční matice udávají délku hlavní a vedlejší osy rozložení "hmoty"v objektu. V našem případě jasové funkce v objektu. Pro zjednodušení si představme, že se jedná o elipsu, kterou objektu opíšeme 13. Je zřejmé, že objekt čtverce a hvězdičky bude mít tyto osy podobně velké, kdežto u obdélníku budou tyto hodnoty rozdílné.



Obrázek 13: Znázornění významu elipticity

Vlastní čísla této matice spočítáme jednoduše podle vzorce

$$\det\begin{pmatrix} \mu'_{2,0}-\lambda & \mu'_{1,1} \\ \mu'_{1,1} & \mu'_{0,2}-\lambda \end{pmatrix} = \left(\mu'_{2,0}-\lambda\right)\left(\mu'_{0,2}-\lambda\right) - \mu'^2_{1,1} = \\ \lambda^2 - \left(\mu'_{2,0}+\mu'_{0,2}\right)\lambda + \left(\mu'_{2,0}\mu'_{0,2} - \mu'^2_{1,1}\right) = 0 \tag{23}$$

Po několika jednoduchých matemetických úpravách rovnice 23 získáme vlastní čísla matice 22

$$a = 1$$
$$b = -\left(\mu'_{2,0} + \mu' 0,2\right)$$
$$c = \left(\mu'_{2,0}\mu_{0,2} - \mu'^2_{1,1}\right)$$
$$D = \left(\mu'_{2,0} - \mu'_{0,2}\right)^2 + 4\mu'^2_{1,1}$$

takže

$$\lambda = \frac{1}{2}\left(\mu'_{0,2} + \mu'_{2,0}\right) \pm \frac{1}{2}\sqrt{\left(\mu'_{0,2} + \mu'_{2,0}\right)^2 + 4\mu'^2_{1,1}} \tag{24}$$

Nyní spočítáme hodnoty parametrů v trénovacím Obrázku 12. Výsledky jsou uvedeny v Tabulce 3. Při sestavení matice pro Random forest zvolíme substituci.

	objekt	$m_{0,0}$	p	λ_{min}	λ_{max}	$\lambda_{min}/\lambda_{max}$	$p^2/m_{0,0}$
1	čtverec	1408	113	114.105	121.325	0.940	9.069
2	čtverec	1393	116	114.268	119.271	0.958	9.66
3	čtverec	1412	113	116.479	120.963	0.963	9.043
4	čtverec	1431	115	116.782	122.121	0.956	9.242
5	hvězdička	435	150	60.898	63.636	0.957	51.724
6	hvězdička	434	155	63.022	63.748	0.989	55.357
7	hvězdička	434	149	60.167	62.501	0.963	51.154
8	hvězdička	432	150	58.433	64.375	0.908	52.083
9	obdélník	699	90	16.531	206.956	0.08	11.588
10	obdélník	739	80	18.076	216.197	0.084	8.660
11	obdélník	713	76	16.472	216.094	0.076	8.101
12	obdélník	700	76	16.5	208.5	0.079	8.251

Tabulka 3: Hodnoty parametrů pro jednotlivé objekty

Nechť čtverec je objekt A, hvězdička objekt B a obdélník objekt C a $F_1 = \lambda_{min}/\lambda_{max}$ a $F_2 = p^2/m_{0,0}$, pak množinu S nazýváme trénovací množinou.

$$S = \begin{pmatrix} F_1 & F_2 & \text{class} \\ 0.940 & 9.069 & A \\ 0.958 & 9.66 & A \\ 0.963 & 9.043 & A \\ 0.956 & 9.242 & A \\ 0.957 & 51.724 & B \\ 0.989 & 55.357 & B \\ 0.963 & 51.154 & B \\ 0.908 & 52.083 & B \\ 0.08 & 11.588 & C \\ 0.084 & 8.660 & C \\ 0.076 & 8.101 & C \\ 0.079 & 8.251 & C \end{pmatrix} \tag{25}$$

Z trénovací množiny vybereme náhodně několik podmnožin. V tomto případě jsem zvolil následující 4 podmožiny. Všimněte si, že není potřeba, aby každá podmnožina obsahovala

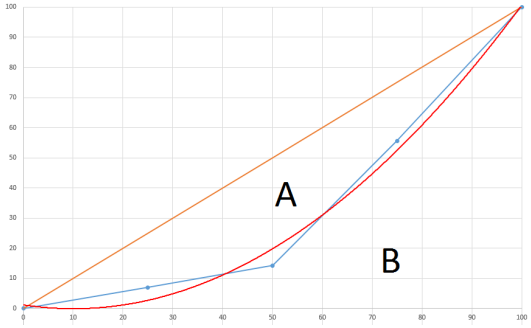
všechny klasifikační třídy.

$$S_1 = \begin{pmatrix} F_1 & F_2 & \text{class} \\ 0.940 & 9.069 & A \\ 0.957 & 51.724 & B \\ 0.989 & 55.357 & B \\ 0.084 & 8.660 & C \end{pmatrix}, \quad S_2 = \begin{pmatrix} F_1 & F_2 & \text{class} \\ 0.940 & 9.069 & A \\ 0.956 & 9.242 & A \\ 0.957 & 51.724 & B \\ 0.908 & 52.083 & B \end{pmatrix}$$
$$S_3 = \begin{pmatrix} F_1 & F_2 & \text{class} \\ 0.963 & 51.154 & B \\ 0.908 & 52.083 & B \\ 0.076 & 8.101 & C \\ 0.079 & 8.251 & C \end{pmatrix}, \quad S_4 = \begin{pmatrix} F_1 & F_2 & \text{class} \\ 0.958 & 9.66 & A \\ 0.963 & 9.043 & A \\ 0.963 & 51.154 & B \\ 0.076 & 8.101 & C \end{pmatrix} \tag{26}$$

Na základě těchto hodnot sestavíme rozhodovací stromy. Při stestavování postupujeme pomocí ID3 algoritmu uvedeného v Kapitole 5.2. Jako divezrifikační parametr jsem použil giniho koeficient. Giniho koeficient definujeme jako obsah plochy mezi Lorenzovou křivkou a diagonálou jednotkového čtverce ku celkové ploše pod diagonálou.

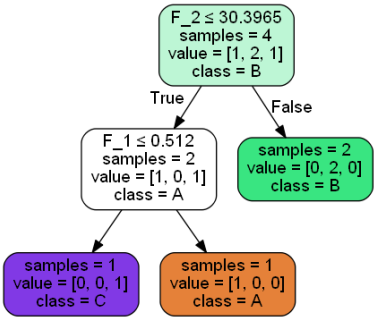
$$GC = \frac{A}{A + B} \tag{27}$$

Nákres situace pro matici $S1$ je uveden na Obrázku 14.

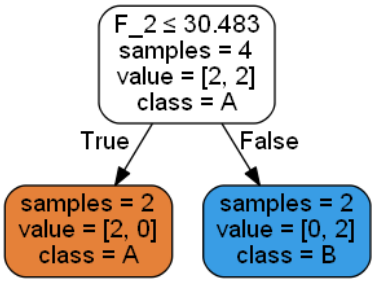


Obrázek 14: Výpočet giniho koeficientu

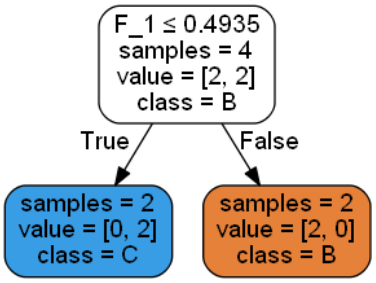
Výsledek je uveden na Obrázcích 15a, 15b, 15c, 15d. Této množině rozhodovacích stromů říkáme Random forest.



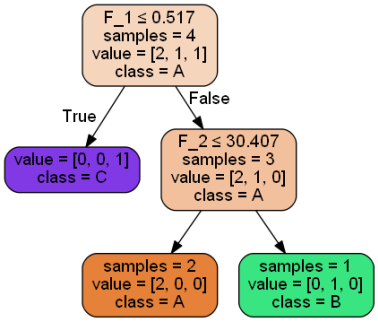
(a) Rozhodovací strom S1



(b) Rozhodovací strom S2



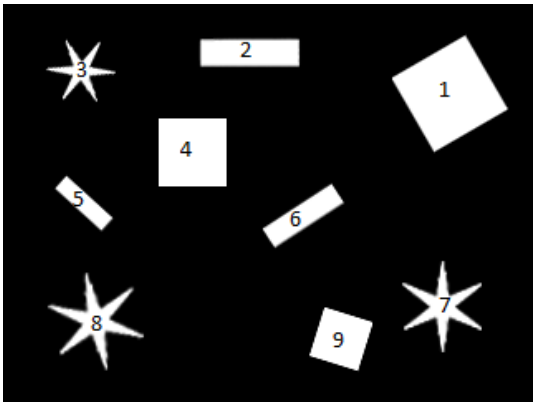
(c) Rozhodovací strom S2



(d) Rozhodovací strom S2

Obrázek 15: Random forest

Nyní pomocí Random forestu uvedeného na Obrázku 15 budeme klasifikovat objekty v testovacím obraze. V testovacím Obraze 16 spočítáme hodnoty F_1 a F_2 a uložíme do matice T (28). Následně vyhodnotíme tyto parametry pro každý strom v náhodném lese.



Obrázek 16: Testovací obrázek

$$T = \begin{pmatrix} & F_1 & F_2 \\ 1 & 0.994022 & 9.40264 \\ 2 & 0.0800296 & 7.97244 \\ 3 & 0.95834 & 51.9894 \\ 4 & 1 & 8.7097 \\ 5 & 0.0751434 & 11.3143 \\ 6 & 0.0745492 & 11.8578 \\ 7 & 0.925664 & 55.5109 \\ 8 & 0.955025 & 55.5738 \\ 9 & 0.992961 & 9.36017 \end{pmatrix} \tag{28}$$

$$P = \left(\begin{array}{c|cc|cccc} & F_1 & F_2 & S_1 & S_2 & S_3 & S_4 \\ 1 & 0.994022 & 9.40264 & A & A & B & A \\ 2 & 0.0800296 & 7.97244 & C & A & C & C \\ 3 & 0.95834 & 51.9894 & B & B & B & B \\ 4 & 1 & 8.7097 & A & A & B & A \\ 5 & 0.0751434 & 11.3143 & C & A & C & C \\ 6 & 0.0745492 & 11.8578 & C & A & C & C \\ 7 & 0.925664 & 55.5109 & B & B & B & B \\ 8 & 0.955025 & 55.5738 & B & B & B & B \\ 9 & 0.992961 & 9.36017 & A & A & B & A \end{array}\right) \tag{29}$$

Když se podíváme na výsledky uvedené v matici T a zaneseme se do výsledkové Tabulky 4, zjistíme, že náš jednoduchý Random forest je schopen vcelku dobře rozhodnout a kategorizovat daný problém.

	objekt	*A*	*B*	*C*
1	čtverec	75%	25%	0%
2	obdélník	25%	0%	75%
3	hvězdička	0%	100%	0%
4	čtverec	75%	25%	0%
5	obdélník	25%	0%	75%
6	obdélník	25%	0%	75%
7	hvězdička	0%	100%	0%
8	hvězdička	0%	100%	0%
9	čtverec	75%	25%	0%

Tabulka 4: Výsledná tabulka

■

7 Vytvoření aplikace pro rozpoznání vozidel v OpenCV

V následující kapitole si popíšeme, jak se dá vytvořit jednoduchý algoritmus pro rozpoznávání vozidel v OpenCv pomocí Random forestu. Jak již název napovídá, k vytvoření aplikace použijeme knihovnu OpenCv, která má již množství metod pro práci s obrazem předimplementovaných. Zejména se bude jednat o metody pro vytvoření HOG a Random forestu. V jednotlivých sekcích si popíšeme základní vstupní hodnoty pro jednotlivé metody. Praktická část, určení vhodných parametrů, se provádí pozorováním a vhodnou úpravou parametrů.

Nejdříve se zaměříme, jakým způsobem je možné popsat vlastnosti obrazové funkce. Jako ukázku jsem zvolil metodu HOG. Následně přejdeme k samotnému vytvoření Random forestu.

7.1 Metoda HOG v OpenCv

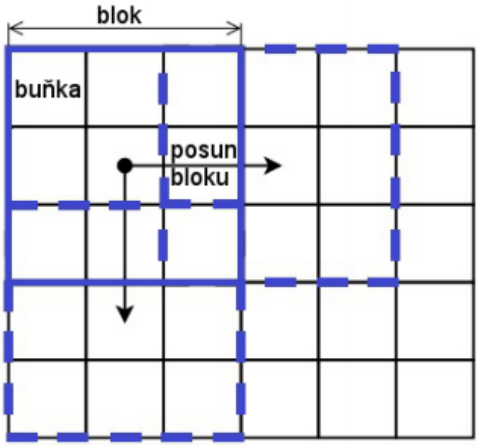
Vytvoření HOG (Histograms of Oriented Gradients) provedeme pomocí třídy cv::HOGDescriptor. Základní vstupní parametry jsou winSize, blockSize, blockStride, cellSize, nbins. Jejich matematické významy jsme si již vysvětlili v Kapitole 3.3. Zde uvedu jen jejich význam a hrubé nastínění, jak se s těmito parametry pracuje:

- **winSize** (*Size*) představuje velikost celého okna, přes které HOG počítáme,
- **blockSize** (*Size*) je velikost bloku, do kterých jsme okno rozdělili,
- **blockStride** (*int*) znamená počet pixelů, o který blok při výpočtu jednotlivých gradientů posouváme,
- **cellSize** (*Size*) je velikost buňky, pro kterou gradient počítáme ,
- **nbins** (*int*) představuje počet výsečí, ve kterých počítáme přírůstek gradientu.

Nástínění toho jak HOG deskriptor provádí výpočet je uvedeno na Obrázku 17. Zároveň je potřeba si uvědomit, že OpenCV vyžaduje validaci

$$\begin{gathered}(\text{winSize.width} - \text{blockSize.width})\%\text{blockStride.width} == 0 \\ \text{and} \\ (\text{winSize.height} - \text{blockSize.height})\%\text{blockStride.height} == 0.\end{gathered} \tag{30}$$

Přeloženo do obecného jazyka je význam předchozí podmínky následovný. Posouváním bloku o blockStride se nám nemůže stát, že budeme počítat hodnotu, která není v obrázku.



Obrázek 17: Princip konstrukce HOG Deskriptoru [11]

7.2 Random forest v OpenCV

Vytvoření Random forestu v OpenCV se porovádí pomocí již implemetovaných metod. V prvé řade existuje funkce **cv::ml::RTrees::create(const cv::ml::RTrees::Params& params=Params())**

Tato metoda obsahuje třídu parametrů cv::ml::RTrees::Params, která se sestává z několika parametrů. Mezi tyto parametry patří například **maxDepth**, **minSampleCount**, **regressionAccuracy**, **useSurrogates**, **maxCategories**, **priors**, **calcVarImportance**, **nactiveVars**, **termCrit**. Význam jednotlivých parametrů je zřejmý už z jejich názvu:

- **maxDepth** (*int*) představuje hodnotu maximální hloubky stromu. (Pokud je zvolena příliš malá hodnota bude rozhodovací schopnost stromu výrazně omezena, naopak bude -li zvolena hodnota příliš velká bude strom příliš ”chytrý”a bude se zbytečně zabývat podrobnostmi, které nejsou relevantní. Informace, které mohou obraz zkreslit patří například šum, rozlišení a jim podobné). Celková velikost stromu může být i menší, pokud bude aplikováno jiné ukončovací kritérium (viz. níže)
- **minSampleCount** (*int*) je minimum vzorků potřebných k dalšímu dělení stromu. (Pokud zvolíme hodnotu tohoto parametru příliš velkou, budou generovány ne-

doučené stromy, naopak pokud bude hodnota příliš malá bude strom přeučený). Vhodná hodnota je malé procento z celkových dat (okolo 1%)

- **regressionAccuracy** (*double*) představuje kritérium pro přerušení při použití regresních stromů. Pokud jsou všechny absolutní diference mezi odhadovanými hodnotami v uzlu a hodnotami z trénovací množiny menší než hodnota tohoto parametru, pak se strom nebude již dále dělit

- **useSurrogates** (*bool*) pokud je nastavena hodnota na ¨*true*, budou vytvářeny náhrady uzlů. Tento parametr je výhodný, pokud pracujeme s neúplnými daty. Proměnná důležitosti bude v tomto případě vypočítána správně

- **maxCategories** (*int*) pokud program ve svém výpočtu vyhodnotí, že se dá množina rozdělit do více kategorií než je nastavená hodnota tohoto parametru, pak najde suboptimální rozdělení do katategorií určených tímto parametrem. Tento parametr je opět výhodný, aby nevznikal zbytečně příliš rozvětvený strom. Algoritmická složitost stromu je exponenciální, takže pokud povolíme dělení do příliš mnoha kategorií, bude výpočet trvat příliš dlouho a spotřebuje příliš mnoho systémových prostředků (například paměti)

- **priors** (*Mat*&) tento parametr ovlivňuje pravděpodobnost výskytu jevu v množině. Pokud pracujeme s daty, kde je minimální výskyt jistého jevu, pak tuto pravděpodobnost utlumíme a dopředu budeme počítat s tím, že se jedná o chybu ve vyhodnocení

- **calcVarImportance** (*bool*) pokud je nastavena hodnota **true**, bude tato hodnota spočítána a je možné ji získat pomocí funkce cv::ml::RTrees::getVarImportance()

- **nactiveVars** (*int*) nastavení velikosti náhodně vybrané podmnožiny vlastností k nalezení nejlepšího rozdělení uzlu stromu do dceřiných uzlů stromu. Pokud je hodnota nastavena na nulu, bude použita defaultní hodnota odmocnina z množství vlastností

- **termCrit** (*cv::TermCriteria*) ukončovací kritéria pro další dělení uzlu při učení. cv::TermCriteria mohou být typu COUNT, EPS nebo COUNT + EPS. Kde COUNT znamená maximální počet iterací nebo elementů a EPS požadovaná přesnost nebo absolutní rozdíl mezi následnými kroky. Často se stává, že trénovací algoritmus se asymptoticky přibližuje k nějaké hodnotě, které nemůže dosáhnout. Proto je vhodné nastavit vhodně tento parametr, aby bylo zamezeno zbytečně složitému počítání již zřejmých hodnot

[3]

7.3 Metoda SVM v OpenCV

V této kapitole je popsáno použití metody SVM v OpenCV. Knihovna OpenCV již obsahuje potřebné algoritmy k vytvoření SVM. Nyní si vysvětlíme jednotlivé parametry metody a jejich význam. Pro vytvoření klasifikátoru SVM využijeme funkci cv::ml::SVM::create(). Mezi základní parametry této metody patří například Type, Kernel, TermCriteria.

- **Type** (*int*) Typ formulace SVM klasifikátoru.
  - **CvSVM::C_SVC** *C*-Support Vector Classification. n-class klasifikace ($n \geq 2$), umožňuje nedokonalé oddělení tříd. Jako diskriminační parametr je použita hodnota *CValue*
  - **CvSVM::NU_SVC** ν-Support Vector Classification. n-class klasificace umožňuje nedokonalé oddělení tříd. Parametr $\nu \in \langle 0, 1 \rangle$. Čím je větší hodnota ν, tím je hranice mezi množinami hladší.
  - **CvSVM::ONE_CLASS** Rozdělení množiny odhadem. Všechna trénovací data pochází z jedné třídy. SVM vytvoří hranici na základě oddělení této třídy od všech případných dalších.
- **CValue** (*double*) Parametr *C* je použit jako diskriminační hodnota v SVM optimalizační úloze C_SVC.
- **Nu** (*double*) Parametr $\nu \in \langle 0, 1 \rangle$ je v SVM optimalizační úloze NU_SVC nebo ONE_CLASS. Čím je větší hodnota ν, tím je hranice mezi množinami hladší.
- **Kernel** (*int*) způsob transformace do prostoru s vyšší dimenzí. Volba tohoto parametru ovlivňuje další povinné parametry
  - **CvSVM::LINEAR** Lineární kernel. Neprovádí se žádné mapování. Lineární klasifikace nebo regrese se provádí v originálním prostoru. $K(x_i, x_j) = x_i^T x_j$
  - **CvSVM::POLY** Polynomiální kernel: $K(x_i, x_j) = (\gamma x_i^T x_j + \text{coef0})^{\text{degree}}$, kde $\gamma > 0$.
  - **CvSVM::RBF** Radial basis function (RBF), vhodná volba pro většinu případů. $K(x_i, x_j) = e^{-\gamma \|x_i - x_j\|^2}$, kde $\gamma > 0$.
  - **CvSVM::SIGMOID** Sigmoid kernel: $K(x_i, x_j) = \tanh(\gamma x_i^T x_j + \text{coef0})$.
- **degree** (*double*) je parametr *degree* použitý pro výpočet kernelu POLY.

- **gamma** (*double*) je parametr γ použitý pro výpočet kernelu v POLY, RBF a SIGMOID.
- **coef0** (*double*) je parameter $coef0$ použitý pro výpočet kernelu v POLY a SIGMOID.
- **class_weights** (*Mat&*) znamená váhu jednotlivých tříd. Parametry C pro jednotlivé třídy jsou získány jako $class\_weights_i * C$. Pomocí tohoto parametru se dají prioritizovat výsledky pro určité třídy.
- **term_crit** (*cv::TermCriteria*) je nastavení ukončovacích kritérií pro trénování SVM.

[3]

7.4 Vytvoření programu

V této kapitole je popis tvorby jednoduchého rozpoznávače vozidel. Nejprve je potřeba připravit a popsat trénovací a testovací množinu. Trénovací množina se musí skládat z dostatečného množství pozitivních a negativních obrazů. Na testovací množině jsme pak schopni zjistit jak dobře náš algoritmus funguje. Následně je potřeba obrazová data popsat (zde je využita metoda HOG) a nakonec naučit program rozpoznávat jednotlivé objekty.

7.4.1 Sestavení trénovací množiny

Pozitivní trénovací množinou je v našem případě soubor 1317 obrazů vozidel o rozměru 128 x 128 px. Příklad několika trénovacích obrazů je uveden na Obrázku č. 18. Jako negativní množinu lze použít jakýkoli obraz, na kterém vozidlo není zobrazeno. V našem případě se negativní množina skládá z 3921 obrazů.

Obrázek 18: Trénovací množina positive

Trénovací vektor sestavíme pomocí metody compute_hog(), která je uvedena v algoritmu 3. Tato metoda spočítá cv::HOGDescriptor pro vektor obrazů img_lst a uloží jeho hodnoty do vektoru gradient_lst. Každý obrázek ve vektoru img_lst je převeden do stupňů šedi. Následně je pro obrázek sestaven vlastní popisný vektor gradientů, který je uložen do gradient_lst. Nastavení velikosti bloku, posunutí bloku, velikosti buňky a počet směrů (binů) provedeme pomocí globálních proměnných, protože při testování obrázků musíme použít stejné hodnoty jako při trénování. Jinak by rozhodovací algoritmus nefungoval.

Pro účely testování a trénování byly zvoleny následující parametry

- CELL_SIZE = Size(8, 8)
- BLOCK_SIZE = Size(16, 16)
- BLOCK_STRIDE = Size(4, 4)
- NBINS = 6

```
void compute_hog(const vector< Mat > & img_lst, vector< Mat > & gradient_lst, const Size & size)
{
    HOGDescriptor hog;
    hog.winSize = size;
    hog.cellSize = CELL_SIZE;
    hog.blockSize = BLOCK_SIZE;
    hog.blockStride = BLOCK_STRIDE;
    hog.nbins = NBINS;
```

```
    Mat gray;

    vector< Point > location;
    vector< float > descriptors;

    vector< Mat >::const_iterator img = img_lst.begin();
    vector< Mat >::const_iterator end = img_lst.end();
    for (; img != end; ++img)
    {
        cvtColor(*img, gray, COLOR_BGR2GRAY);
        equalizeHist( gray, gray );
        hog.compute(gray, descriptors);
          Mat g;
          Mat(descriptors).convertTo(g, CV_32F);
         gradient_lst.push_back(g);
    }
}
```

Výpis 3: Metoda pro získání trénovacích vektorů

7.4.2 Vytvoření Random forestu

Jakmile jsou načtena všechna data z trénovací množiny vložíme je do struktury cv::Ptr<cv::ml::TrainData>, kterou následně využijeme při vytváření Random forestu. Zde je důležité mít uloženo který záznam patří do které skupiny dat. V našem případě máme ve vektoru labels uloženy skupiny 1 pozitivní a 0 negativní, které odpovídají pořadí načítaných objektů. Zdrojový kód je uveden ve výpise 4.

V posledním kroku trénování vytvoříme Random Forest, který uložíme do struktury cv::Ptr<cv::ml::RTrees>. Volbu parametrů provádíme pozorováním chování rozhodovacího algoritmu a vhodnou korekcí jednotlivých parametrů.

```
Ptr<TrainData> prepareTrainData(vector< Mat > & gradient_lst, vector<int> & labels)
{
    Mat samples(gradient_lst.size(), gradient_lst[0].rows,CV_32F);

    int col = 0;
    for(int i=0; i < samples.rows; i++)
    {
        for(int j=0; j < samples.cols; j++)
        {
            samples.at<float>(i,j) = gradient_lst.at(i).at<float>(col,j);
        }
```

```
    }
    return TrainData::create(samples, 0, Mat(labels));
}

Ptr<RTrees> trainRTrees(Ptr<TrainData> & train_data)
{
    Ptr<RTrees> rtrees = RTrees::create();
    rtrees->setMaxDepth(50);
    rtrees->setMinSampleCount(600);
    rtrees->setActiveVarCount(0);
    rtrees->setTermCriteria(TermCriteria(TermCriteria::EPS + TermCriteria::MAX_ITER, 1000,
        0.05));

    rtrees->train(train_data);

    return rtrees;
}
```

Výpis 4: Příprava trénovací dat a sestavení Random Forestu

7.4.3 Vytvoření SVM

Pro načtení trénovacích dat do struktury cv::Ptr<cv::ml::TrainData>využijeme funkci prepareTrainData (Výpis 4) z předchozí Podkapitoly 7.4.2

```
Ptr<SVM> trainSVM(Ptr<TrainData> & train_data, int Kernel)
{
    double C;

    Ptr<SVM> svm = SVM::create();
    svm->setType(SVM::C_SVC);
    svm->setKernel(Kernel);

    if (Kernel == SVM::RBF)
    {
        C = 10;
        svm->setGamma(0.005);
    }
    else if (Kernel == SVM::LINEAR)
    {
        C = 0.01;
    }
```

```
    svm->setTermCriteria(TermCriteria(TermCriteria::EPS + TermCriteria::MAX_ITER, 1000, 0.05));
    svm->setC(C);
    svm->train(train_data);

    return svm;
}
```

Výpis 5: Sestavení SVM

7.5 Vyhledávání objektu v reálném obraze

Aby bylo možné tuto metodu aplikovat v praxi, je zapotřebí algoritmus rozšířit tak, aby byl schopen rozpoznat osobní vozidlo i v obraze, který má jiný formát než 128 x 128 px.

Vyhledávání provádíme cyklickým procházením objektu pomocí vyhledávacího okna tzv. ROI (Region Of Interest) o rozměru 128 x 128 px. Každý tento výřez necháme vyhodnotit Random forest nebo SVM, abychom zjistili, jestli se jedná o vozidlo nebo ne.

Aby bylo možné porovnávat obrazy různých velikostí, zvolil jsem metodu zmenšování obrazu. V prvním průchodu maska pro výpočet prochází přes původní velikost obrazu a při každém dalším se celý obraz zmenší na 80% své aktuální velikosti. Obraz je zmenšován do té doby, dokud je jeho výška i šířka větší než 128 x 128 px. Zdrojový kód této funkce je uveden ve Výpise 6.

Zde také použijeme další dvě globální proměnné

- WINDOW_SIZE = Size(128, 128)
- WINDOW_STEP = 32

```
vector<vector<float> > testNormalImage(Mat &img, Ptr<RTrees> & rTrees)
{
    HOGDescriptor hog;
    hog.winSize = WINDOW_SIZE;
    hog.cellSize = CELL_SIZE;
    hog.blockSize = BLOCK_SIZE;
    hog.blockStride = BLOCK_STRIDE;
    hog.nbins = NBINS;

    vector< float > descriptors;
    Mat g;
```

```
vector<vector<float> > result;
Mat gray;
cvtColor(img, gray, COLOR_BGR2GRAY);
Mat rescaleImg = gray.clone();

float resizeFactor = 0.8;
float factor = 1;

while(rescaleImg.rows - WINDOW_SIZE.height > 0 && rescaleImg.cols - WINDOW_SIZE.
    width > 0)
{
    for (int row = 0; row <= rescaleImg.rows - WINDOW_SIZE.height; row +=
        WINDOW_STEP)
    {
        for (int col = 0; col <= rescaleImg.cols - WINDOW_SIZE.width; col +=
            WINDOW_STEP)
        {
            Rect windows(col, row, WINDOW_SIZE.height, WINDOW_SIZE.width);
            Mat countImg = rescaleImg.clone();
            rectangle(countImg, windows, Scalar(255), 1, 8, 0);

            Mat Roi = countImg(windows);
            equalizeHist( Roi, Roi );

            hog.compute(Roi, descriptors);

            Mat(descriptors).convertTo(g, CV_32F);

            int predict = rTrees->predict(g);

            if (predict == 1)
            {
                vector<float> r;
                r.push_back(factor);
                r.push_back(col);
                r.push_back(row);
                result.push_back(r);
            }

            Roi.release();
            countImg.release();
        }
    }
    resize(rescaleImg,rescaleImg,Size(),resizeFactor,resizeFactor);
    factor *= resizeFactor;
```

```
        }

        rescaleImg.release();
        gray.release();
        g.release();
        vector<float>().swap(descriptors);


        return result;
    }
```

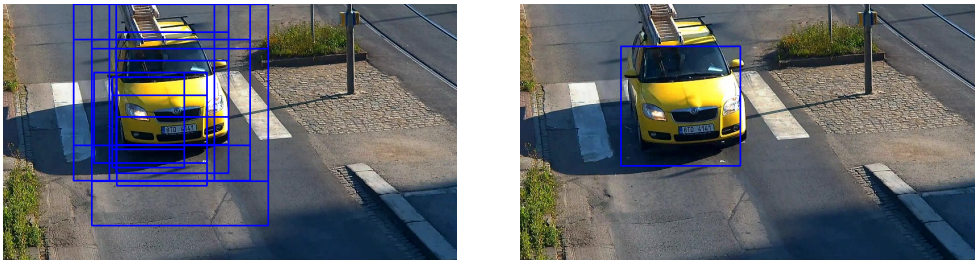
Výpis 6: Testování reálného obrazu

Tato metoda nám vrátí velké množství pozitivních výskytů, které většinou překrývají objekt. Ukázka pozitivní detekce je uvedena na obrázku 19a. Eliminaci tohoto jevu provedeme pomocí metody cv::groupRectangles(std::vector¡Rect¿& rectList, int groupThreshold, double eps = 0.2). Parametr groupThreshold udává počet překrývajících se čtverců, aby byl výskyt považován za pozitivní. Parametr eps nastavuje relativní rozdíl mezi čtverci, aby byly ještě považovány za shodné.

Na základě testování a pozorování správnosti vyhodnocení jsem dospěl k tomu, že vhodné nastavení parametrů groupThreshold a eps je pro statistickou kameru

- groupThreshold = 3
- eps = 5

Výsledek je uveden na Obrázku 19b



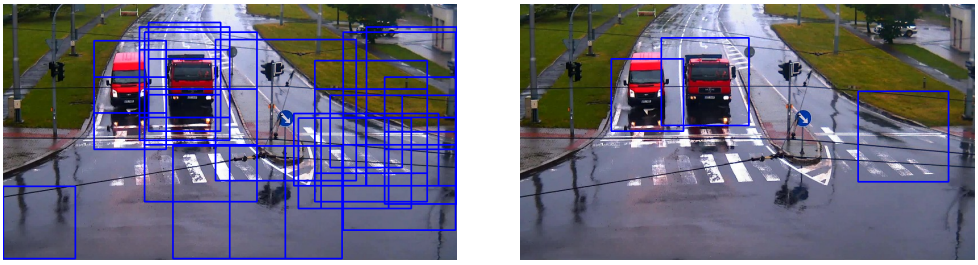
(a) Pozitivní detekce objektu klasifikátorem (b) Korekce metodou groupRectangles

Obrázek 19: Detekce objektu v reálném obraze, statistická kamera

Pro zpracování obrazů z přehledové kamery vyšly nejlépe parametry

- groupThreshold = 1
- eps = 0.5

Výsledek je uveden na Obrázku 20b



(a) Pozitivní detekce objektu klasifikátorem (b) Korekce metodou groupRectangles

Obrázek 20: Detekce objektu v reálném obraze, přehledová kamera

8 Porovnání metod Random forest a SVM

V této kapitole se budeme zabývat srovnáním klasifikačních algoritmů Random forest a SVM s kernelem RBF a kernelem Linear. V první části se zaměříme na optimální nastavení parametrů. V druhé části provedeme měření úspěšnosti klasifikátorů na reálných obrazech. Ve třetí části se zaměříme na časy potřebné pro výpočet jednotlivých metod.

8.1 Optimální nastavení parametrů

Nastavení parametrů metody ovlivňuje její schopnost klasifikovat. Při chybném nastavení dochází k přeučení nebo nedoučení algoritmu, případně k velké benevolenci chybných detekcí. Jako postup jsem zvolil cyklickou změnu parametrů. Pro učení byla využita databáze 1317 pozitivních a 3921 negativní obrazů o rozměru 128 x 128 px. Následně byla natrénovaná metoda podrobena testu na 500 pozitivních a 500 negativních obrazech o rozměrech 128 x 128 px.

8.1.1 Random forest

U metody Random forest bylo testováno nastavení parametrů maxDepth (maximální hloubka stromu) a minSampleCount (minimální počet bodů v uzlu potřebných pro další dělení). Pro trénování a testování byly použity parametry MaxDepth $\in \langle 50, 1000 \rangle$ s krokem 50 a MinSampleCount $\in \langle 50, 1000 \rangle$ s krokem 50. Optimální nastavení parametrů je uvedeno v tabulce 5. Zkratka MSCnt znamená parametr MinSampleCount.

MaxDepth	MSCnt	TP	TN	FP	FN	Přesnost	TPR	FPR	PPV	NPV
50	600	456	499	1	44	0.955	0.912	0.002	0.998	0.919
⋮					⋮					⋮
500	600	456	499	1	44	0.955	0.912	0.002	0.998	0.919
⋮					⋮					⋮
1000	600	456	499	1	44	0.955	0.912	0.002	0.998	0.919

Tabulka 5: Nastavení parametrů Random forest

Z výsledků je patrné, že při nastavení prarametru MinSampleCount = 600 nemá nastavení MaxDepth $\in \langle 50, 1000 \rangle$ s krokem 50 na chybu klasifikace vliv.

V praktické části jsem tedy využil parametr MaxDepth = 50 a MinSampleCount = 600.

8.1.2 SVM RBF

U metody SVM s kernelem RBF bylo testováno nastavení parametrů γ a C.

Parametr γ definujeme intuitivně jako: Jak daleko dopadá vliv jediného trénovacího případu. Malé hodnoty parametru γ znamenají velký vliv jednotlivého případu, naopak velké hodnoty znamenají malý vliv. Parametr γ tedy lze považovat jako inverzi k poloměru vlivu jednotlivých vzorků vybraných modelem jako podpůrné vektory.

Parametr C eliminuje chybné zařazení tréningových vzorků vzhledem k jednoduchosti rozhodovacího povrchu. Malá hodnota parametru C vytváří hladkou plochu, zatímco vysoká hodnota parametru C má za cíl správně klasifikovat všechny trénovací obrazy. Tím parametr C umožňuje volnost výběru více vzorků jako podpůrných vektorů.

Pro trénování a testování byly použity hodnoty parametru C = 10^n, kde $n \in \langle -5, 5\rangle \cap \mathbb{Z}$ a $\gamma = \frac{1}{n}$, kde $n \in \langle 100; 200\rangle$ s krokem 10

C	γ	TP	TN	FP	FN	Přesnost	TPR	FPR	PPV	NPV
10	0.005	498	500	0	2	0.998	0.996	0	1	0.994
⋮					⋮					⋮
100000	0.005	498	500	0	2	0.998	0.996	0	1	0.994

Tabulka 6: Nastavení parametrů SVM RBF

Zde si můžeme všimnout, že pro dané nastavení γ už nastavení parametru C > 10 nemá vliv.

V praktické části tedy bude použito nastavení $\gamma = 0.005$ a C = 10.

8.1.3 SVM Linear

U metody SVM s kernelem Linear bylo testováno nastavení parametru C.

Pro zkoumání úspěšnosti klasifikace byly použity hodnoty $C = 10^n$, kde $n \in \langle -5, 5\rangle \cap \mathbb{Z}$.

C	TP	TN	FP	FN	Přesnost	TPR	FPR	PPV	NPV
0.01	495	497	3	5	0.992	0.990	0.006	0.994	0.990

Tabulka 7: Nastavení parametrů SVM Linear

Na zkoumané množině C vyšlo nejlépe pouze jedno nastavení. Proto bude v dalších částech při metodě SVM, kernel Linear, použito nastavení C = 10.

8.2 Porovnání úspěšnosti klasifikace

Pro trénování algoritmů Random forest, SVM RBF a SVM Linear byla použita množina 1317 pozitivních a 3921 negativní obrazů o rozměru 128 x 128 px.

8.2.1 Obrazy ze statistické kamery

Obrazy ze statistické kamery jsou takové obrazy, kde se na záběru vyskytuje zpravidla právě jedno vozidlo. Záběry z těchto kamer se v praxi používají například na počítání aut, která přes křižovatku projedou.

Testování úspěšnosti proběhlo na 164 obrazech ze statistické kamery o rozměrech 800 x 450 px. Pro nastavení parametrů jednotlivých metod byly použity výsledky uvedené v Kapitole 8.1.

	TP	TN	FP	FN	TPR	FPR	PPV	NPV	F_1
Rtrees	135	83644	11	14	0.9060	0.0001	0.3767	0.9982	0.9153
SVM RBF	141	83651	4	8	0.9463	0.0001	0.9724	0.9999	0.9592
SVM LINEAR	141	83646	9	8	0.9463	0.0001	0.4209	0.9987	0.9431

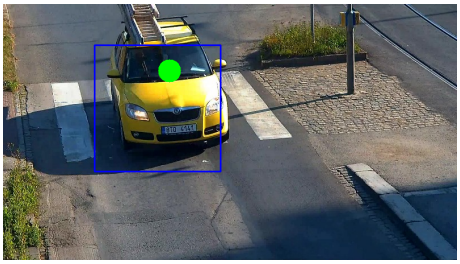
Tabulka 8: Výsledky porovnání úspěšnosti klasifikace, statistická kamera

Výsledky uvedené v Tabulce 8 ukazují, že je metoda SVM, ať už s kernelem RBF nebo Linear v klasifikaci úspěšnější. Nejlepšího výsledku dosáhla metoda SVM s kernelem RBF, jejíchž F_1-Score bylo 0.9592.

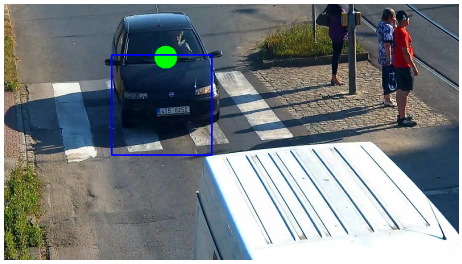
Metoda Random forest v tomto testu dopadla nejhůře. F_1-Score bylo 0.9153.

Na Obrázku 21 jsou zobrazeny příklady klasifikací jednotlivými metodami. Zelený bod určuje pozici auta. Modrý čtverec zobrazuje oblast ve které klasifikátor identifikoval

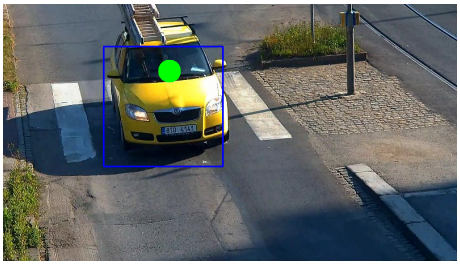
vozidlo. Jako pozitivní výskyt je považován případ, kdy je zelený bod uvnitř modrého čtverce.



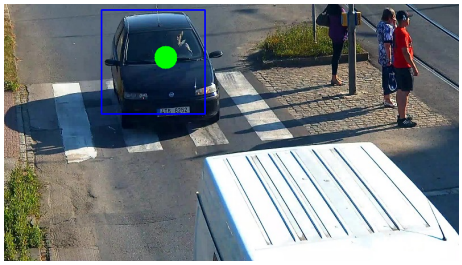
(a) Detekce RF test 1



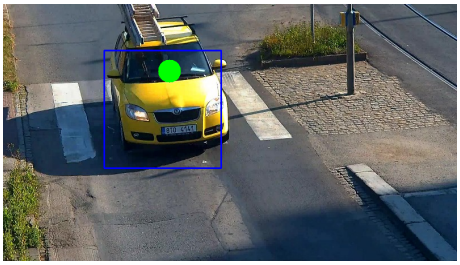
(b) Detekce RF test 2



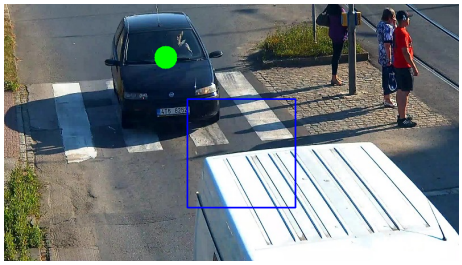
(c) Detekce SVM RBF test 1



(d) Detekce SVM RBF test 2



(e) Detekce SVM Linear test 1



(f) Detekce SVM Linear test 2

Obrázek 21: Ukázka klasifikace, statistická kamera

8.2.2 Obrazy z přehledové kamery

Přehledová kamera snímá situaci na celé křižovatce. Na kameře je tedy zachyceno větší množství aut z různých úhlů.

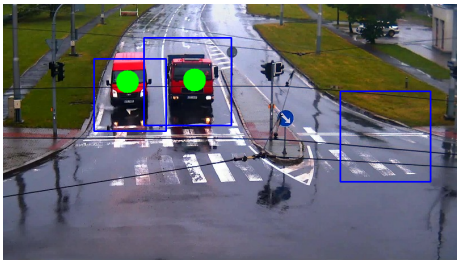
Testování úspěšnosti proběhlo na 152 obrazech z přehledové kamery o rozměrech 800 x 450 px. Pro nastavení parametrů jednotlivých metod byly použity výsledky uvedené v Kapitole 8.1.

	TP	TN	FP	FN	TPR	FPR	PPV	NPV	F_1
Rtrees	162	77104	268	138	0.5400	0.0035	0.3767	0.9982	0.4438
SVM RBF	188	77295	77	112	0.6267	0.0010	0.7094	0.9986	0.6655
SVM LINEAR	197	77101	271	103	0.6567	0.0035	0.4209	0.9987	0.5130

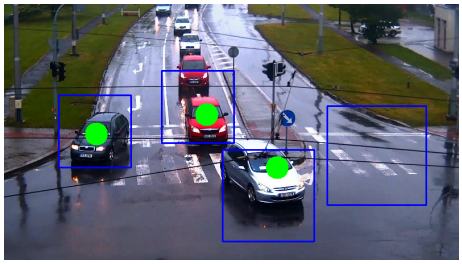
Tabulka 9: Výsledky porovnání úspěšnosti klasifikace, přehledová kamera

Z naměřených hodnot v Tabulce 9 je zřejmé, že úspěšnost klasifikace v tomto případě výrazně klesla oproti výsledkům ze statistické kamery. Tato skutečnost se způsobena tím, že auta jsou snímána z různých úhlů, v záběru kamery jsou troleje tramvaje a křižovatka je snímána za deště. Výsledky by se daly ovlivnit zvětšením trénovací množiny o falešně negativní a falešně pozitivní výsledky.

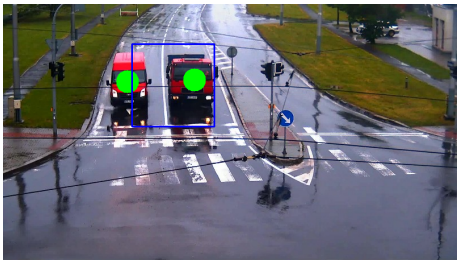
Na Obrázku 22 jsou zobrazeny příklady klasifikací jednotlivými metodami. Zelený bod určuje pozici auta. Modrý čtverec zobrazuje oblast ve které klasifikátor identifikoval vozidlo. Jako pozitivní výskyt je považován případ, kdy je zelený bod uvnitř modrého čtverce.



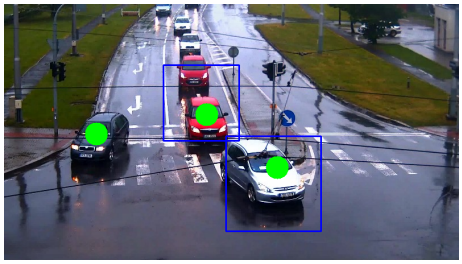
(a) Detekce RF test 1



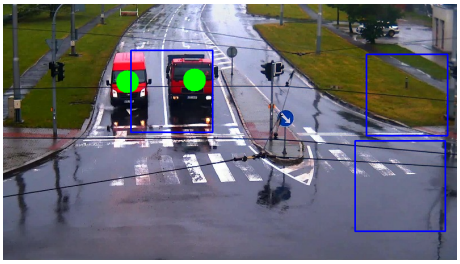
(b) Detekce RF test 2



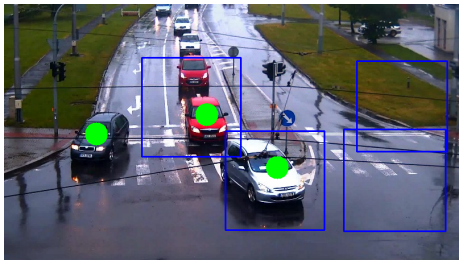
(c) Detekce SVM RBF test 1



(d) Detekce SVM RBF test 2



(e) Detekce SVM Linear test 1



(f) Detekce SVM Linear test 2

Obrázek 22: Ukázka klasifikace, přehledová kamera

8.3 Porovnání časové náročnosti

Při porovnání časové náročnosti jsem se zaměřil na čas trénování a testování. Při trénování může hrát čas zanedbatelnou roli, ale v případě testování je rychlost rozhodování jednou z klíčových vlastností algoritmu. Zvláště chceme -li klasifikovat real time data.

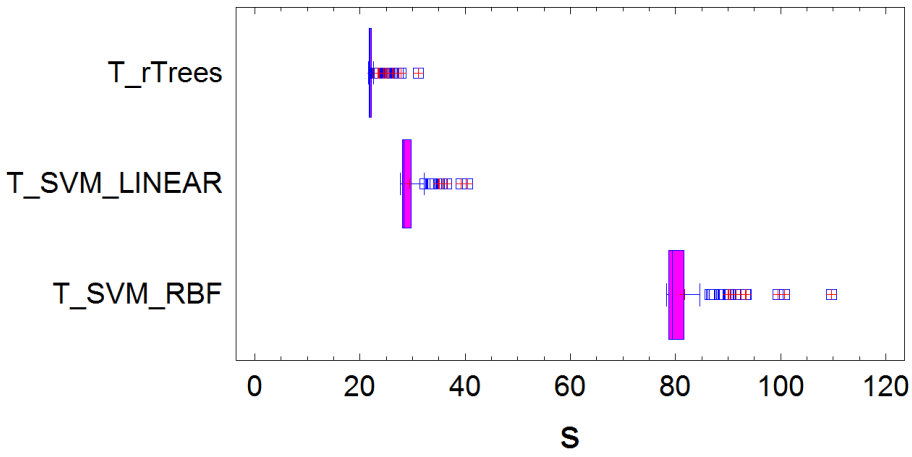
Je potřeba si uvědomit, že v reálné praxi má na dosažené hodnoty vliv také konfigurace počítače. Pro naše účely demonstrativního srovnání doby běhu klasifikačních metod RF a SVM jsou však tyto údaje irelevantní.

8.3.1 Trénování klasifikátorů

Trénování klasifikátorů probíhalo na množině 1317 pozitivních 3921 negativních objektů. Nastavení parametrů bylo provedeno na základě výsledků z kapitoly 8.1. Každá metoda byla podrobena testu při 100 opakováních. Výsledky experimentů jsou uvedeny v tabulce 10 a na grafu 23. V rámci testu jsem se také zaměřil na metodu HOG, které zpracování trénovací množiny trvalo celkem 8.3857 ± 0.4799 s.

	Medián	Rozptyl	Směrodatná odchylka	Minimum	Maximum
T_rTrees	21.9037	2.27086	1.50694	21.5792	31.1635
T_SVM_LINEAR	28.3972	6.1655	2.48304	27.6763	40.5103
T_SVM_RBF	79.4445	29.7731	5.45647	78.2599	109.734

Tabulka 10: Čas trénování



Obrázek 23: Doba trénování klasifikačních metod

Z naměřených hodnot je patrné, že metoda RF je ve srovnání s metodou SVM výrazně rychlejší. Z důvodu, že čas trénování není v praxi většinou rozhodující faktor, mají uvedené hodnoty spíše informativní charakter.

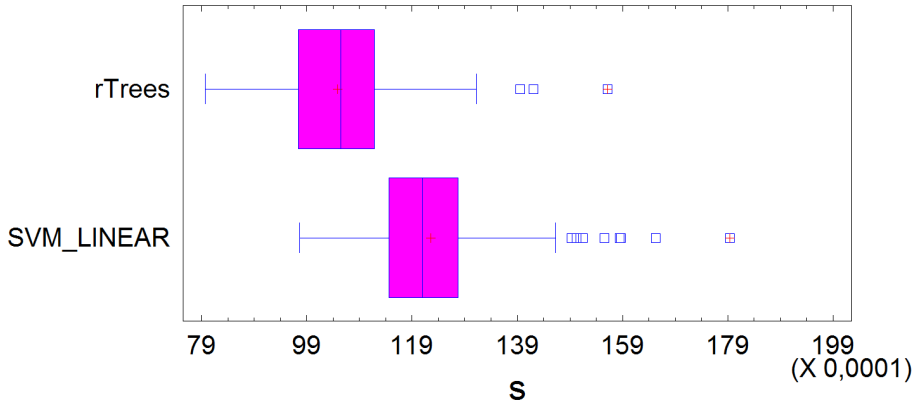
8.3.2 Testování klasifikátorů

Testování klasifikačních algoritmů probíhalo na reálných 164 obrazech o rozměrech 800 x 450 px. Úspěšnost klasifikace je řešena v kapitole 8.2.

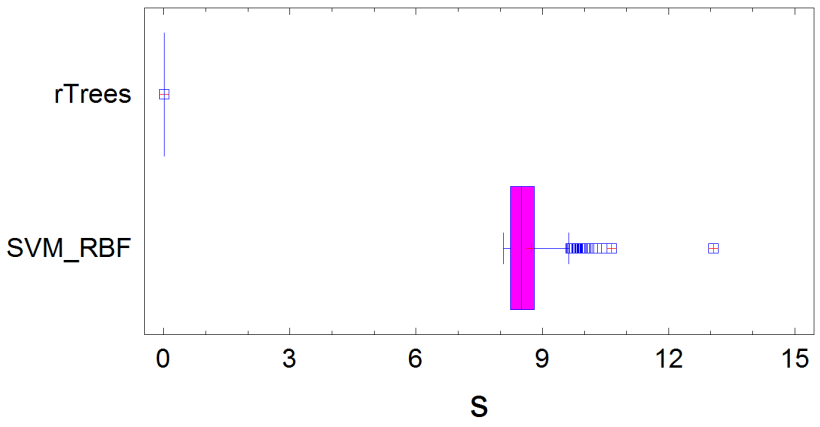
Výsledky uvedené v tabulce 11 a grafech 24 a 25 odpovídají času pro zpracování jednoho obrazu o rozměrech 800 x 450 px. Výpočet HOG pro daný obraz trval 0.8291 ± 0.0435

	Medián	Rozptyl	Směrodatná odchylka	Minimum	Maximum
rTrees	0.0105455	0.000001554	0.00124659	0.007967	0.015615
SVM_LINEAR	0.0121	0.00000159718	0.0012638	0.00976598	0.017938
SVM_RBF	8.49864	0.533218	0.730218	8.07339	13.0683

Tabulka 11: Čas testování



Obrázek 24: Testování RF x SVM Linear



Obrázek 25: Testování RF x SVM RBF

Z naměřených hodnot vyplývá, že je metoda Random forest výrazně rychlejší také v testovací fázi.

Ačkoli byla metoda SVM s kernelem RBF při klasifikaci nejúspěšnější je pro zpracování real time dat při daném nastavení nepoužitelná. Pro zpracování snímku touto metodou je potřeba po započítání kalkulace deskriptoru HOG přibližně 9.5602 ± 0.7737 s. Při použití metody SVM Linear se potřebný čas pro výpočet zkrátí na 0.9518 ± 0.0448 s. Metoda RF dosáhne dokonce času 0.8397 ± 0.0448 s.

9 Závěr

Cílem této práce bylo popsat metodou Random forest a demonstrovat její funkci na praktické aplikaci – rozpoznávání objektů v obraze.

V první části bylo ukázáno jak sestavit Random forest bez použití programových knihoven. Fungování tohoto algoritmu bylo vysvětleno na jednoduchém příkladě rozlišování geometrických tvarů (příklad č. 6.2 Kapitola 6). Tento příklad ukazuje jednoduchost a rychlost při použití této metody.

V další části bylo provedeno testování úspěšnosti klasifikace pomocí metody Random forest a SVM s kernelem RBF a kernelem Linear. V tomto testu byla metoda SVM RBF nejspolehlivější, která z celkového počtu 83804 testů chybně vyhodnotila pouze 12 případů, z nichž byly 4 výskyty falešně pozitivní a 8 falešně negativních. Metoda Random forest vyhodnotila chybně 25 záznamů. 11 testů bylo falešně pozitivních a 14 falešně negativních.

Následně bylo provedeno testování časové náročnosti jednotlivých algoritmu. V tomto testu byla dominantní zase metoda Random forest. Medián doby analýzy obrazu 800 x 450 px (511 testů) metody Random forest byl 0.0105 s. Metoda SVM, kernel RBF, analyzovala daný obraz 8.4986 s.

Výsledkem této práce je program pro rozpoznávání vozidel ve videozáznamu. Na zákaldě testů na časovou náročnost byl algoritmus SVM RBF vyhodnocen jako nevhodný. Ačkoli je ze zkoumaných metod nejspolehlivější, je jeho časová náročnost taková, že se pro zpracování real-time dat nehodí.

10 Reference

[1] MRÁZOVÁ, Iveta. *Dobývání znalostí* [online]. [cit. 2016-02-07]
Dostupné z: http://ksvi.mff.cuni.cz/ mraz/datamining/

[2] RUSSELL, Stuart J. a Peter. NORVIG. *Artificial intelligence: a modern approach*
Englewood Cliffs, N.J.: Prentice Hall, c1995. ISBN 01-310-3805-2.

[3] *OpenCV documentation* [online]. [cit. 2016-02-16] Dostupné z: http://docs.opencv.org/

[4] Random forest. In: *Wikipedia: the free encyclopedia* [online].
San Francisco (CA): Wikimedia Foundation [cit. 2016-02-16].
Dostupné z: https://en.wikipedia.org/wiki/Random_forest

[5] ID3 algorithm. In: *Wikipedia: the free encyclopedia* [online].
San Francisco (CA): Wikimedia Foundation [cit. 2016-03-06].
Dostupné z: https://en.wikipedia.org/wiki/ID3_algorithm

[6] Evaluation of binary classifiers. In: *Wikipedia: the free encyclopedia* [online].
San Francisco (CA): Wikimedia Foundation [cit. 2016-07-06].
Dostupné z: https://en.wikipedia.org/wiki/Evaluation_of_binary_classifiers

[7] Kernel method. In: *Wikipedia: the free encyclopedia* [online].
San Francisco (CA): Wikimedia Foundation [cit. 2016-07-13].
Dostupné z: https://en.wikipedia.org/wiki/Kernel_method

[8] TIN KAM HO. Random decision forests. *Proceedings of 3rd International Conference on Document Analysis and Recognition.* IEEE Comput. Soc. Press, 1995, 278-282.
DOI: 10.1109/ICDAR.1995.598994. ISBN 0-8186-7128-9.
Dostupné také z: http://ieeexplore.ieee.org/document/598994/

[9] MITCHELL, Tom M.*Machine Learning*.New York: McGraw-Hill, c1997.
ISBN 00-704-2807-7.

[10] MAŠEK, Jan. *Detekce objektů v obraze s pomocí Haarových příznaků* [online].
Brno, 2012 [cit. 2017-03-12].
Dostupné z: https://www.vutbr.cz/www_base/zav_prace_soubor_verejne.php?file_id=51400

[11] HRÚZ, Marek. *LBP, HoG* [online]. Plzeň [cit. 2017-04-15]
Dostupné z: http://www.kky.zcu.cz/uploads/courses/mpv/05/materialy05.pdf